Indian Academy of Sciences, Bengaluru
Indian National Science Academy, New Delhi
The National Academy of Sciences India, Prayagraj

SUMMER RESEARCH FELLOWSHIPS — 2024

Format for the final Report *,^

Name of the candidate	:	Nithish Chouti
Application Registration no.	:	ENG83700
Date of joining	:	20th May, 2024
Date of completion	:	18th July, 2024
Total no. of days worked	:	60 days
Name of the guide	:	Prof. Sumohana S. Channappayya
Guide's institution	:	Indian Institute of Technology, Hyderabad.
Project title	:	Semantic Segmentation using Deeplabv3+ with MobileNet Backbones

Address with pin code to which the certificate could be sent:

4-1-102/22/302, HMT HOMES APARTMENT, STR.No-4, BHAVANI NAGAR, NACHARAM, HYDERABAD, TELANGANA, 500076

E-mail ID: nithishchouti2003@gmail.com

Phone No: 9493865924

TA Form attached with final report : YES ✓ NO –

If, NO, Please specify reason –

Ch. Nithish

Signature of the candidate

Date: 18/7/24

Signature of the guide

Date: 18/7/24

IMPORTANT NOTES:

*** This format should be the first page of the report and should be stapled with the main report. The final report could be anywhere between 20 and 25 pages including tables, figures etc.**

^ The final report must reach the Academy office within 10 days of completion. If delayed fellowship amount will not be disbursed.

(For office use only; do not fill/tear)

Candidate's name:	Fellowship amount:
Student: Teacher:	Deduction:
Guide's name:	TA fare:
KVPY Fellow: INSPIRE Fellow:	Amount to be paid:
PFMS Unique Code:	A/c holder's name:
Others	

Summer Internship Report

May-July, 2024

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Title: Semantic Segmentation using DeepLabv3+ with MobileNet Backbones

Nithish Chouti (ENGS3700)

Supervisor
Prof Sumohana S Channappayya,
Dept of Electrical Engineering

Acknowledgement

I would like to express my sincere gratitude to Prof. Sumohana S Channappayya for his invaluable guidance, support, and encouragement throughout this internship. His expertise and mentorship have been instrumental in enhancing my understanding of computer vision and deep learning. I am honoured to be a part of the project **'Developing Vision Capabilities for Bi-Pedaled Robot'** by contributing my work on **'Semantic Segmentation using DeepLabv3+ with MobileNet Backbones'**.

I am also thankful to the entire team at the Department of Electrical Engineering, IIT Hyderabad, for providing me with a stimulating environment to work and learn. Special thanks to Ashwath Sreeram, Athira Krishnan, and Ambarish Parthasarathy for their assistance and valuable insights and discussions.

Additionally, I extend my appreciation to Indian Academy of Sciences, Indian National Science Academy and National Academy of Sciences, whose support made this internship possible.

Finally, I am deeply grateful to my family and friends for their unwavering encouragement and understanding during this journey.

Nithish Chouti
(ENGS3700)

Contents

1 Problem Statement

The project led by Prof. Sumohana S. Channappayya from the Electrical Engineering Department at the Indian Institute of Technology, Hyderabad, focuses on 'Developing Vision Capabilities for Bi-Pedaled Robot'. The research team, comprising Athira, Ambarish, Akshad, Ashwath Sreeram, and other interns, is working on various subtasks of the project such as object detection, image segmentation, change detection, and crowd counting. This report specifically delves into the task of image segmentation under the expertise of Ashwath Sreeram. The problem statements addressed in the scope are:

1. Exploring DeepLabv3+ and Segformer for Semantic Segmentation:
    - Investigate the architectures and potential applications of DeepLabv3+ and Segformer models for semantic segmentation tasks.
2. Evaluating MobileNet Backbones in DeepLabv3+:
    - Assess the performance of MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small backbones when integrated with the DeepLabv3+ architecture for semantic segmentation tasks.
3. Comparative Analysis of Model Efficiency:
    - Conduct a comparative analysis of the efficiency and performance of different MobileNet backbones with DeepLabv3+in terms of accuracy, mIoU, inference time, GPU memory usage, and computational complexity (parameters and FLOPs).

2 Introduction

Semantic segmentation is a critical task in computer vision that involves labeling each pixel in an image with a corresponding class [3]. This pixel-level classification enables detailed understanding of visual scenes, which is crucial for various applications such as autonomous driving, medical imaging, environmental monitoring, etc.

DeepLabv3+ is a state-of-the-art model for semantic segmentation, known for its ability to capture multi-scale context through atrous convolution and refine segmentation results with a sophisticated decoder module. In this project, we explore the performance of DeepLabv3+ with different MobileNet backbones, specifically MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small, to assess their efficiency and effectiveness in semantic segmentation tasks.

The primary objective of this study is to evaluate the trade-offs between model complexity, accuracy, and computational efficiency across these MobileNet variants. By conducting experiments on the PASCAL VOC 2012 and ADE20K datasets, we aim to provide a comprehensive comparison highlighting the strengths and weaknesses of each backbone within the DeepLabv3+ framework. This analysis will offer valuable insights into selecting appropriate models for deployment in resource-constrained environments.

3 Literature review

The field of semantic segmentation has seen significant advancements with the development of various neural network architectures. These architectures range from traditional Convolutional Neural Networks (CNNs) to more sophisticated models incorporating transformers and depthwise separable convolutions.

Fully Convolutional Networks (FCNs) for Semantic Segmentation Fully Convolutional Networks (FCNs) were among the first deep learning models applied to semantic segmentation. FCNs replace fully connected layers with convolutional layers, enabling pixel-wise prediction. The model typically processes RGB images with input dimensions such as 224x224, producing pixel-wise class labels. FCNs have been evaluated on datasets like PASCAL VOC and have relatively high inference times compared to more recent architectures. [4]

DeepLabv3+ for Semantic Image Segmentation DeepLabv3+ is a prominent architecture that combines atrous convolution with a decoder module to refine segmentation results. The encoder leverages a backbone such as ResNet or MobileNet to extract robust features, while the Atrous Spatial Pyramid Pooling (ASPP) module captures multi-scale context. This architecture typically operates on RGB images with input dimensions around 512x512, providing pixel-wise class labels as output. It has been evaluated on datasets like PASCAL VOC 2012 and Cityscapes, with inference times varying based on the backbone used; for instance, ResNet-101 achieves around 0.1 seconds per image. [5]

SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers SegFormer introduces a hierarchical vision transformer backbone that efficiently processes tokens to capture multi-scale features through sequential transformer layers. This architecture processes RGB images, typically at 512x512 dimensions, to produce pixel-wise class labels. SegFormer has shown excellent performance on datasets like ADE20K and Cityscapes, benefiting from the efficiency of its attention mechanisms which generally result in faster inference times compared to traditional convolutional neural networks. [6]

MobileNetV2: Inverted Residuals and Linear Bottlenecks MobileNetV2 is designed to enhance efficiency and performance with its inverted residuals and linear bottlenecks. Utilizing depthwise separable convolutions, this architecture significantly reduces computational cost. It typically processes RGB images at dimensions like 224x224, outputting feature maps that can be used for segmentation or other tasks. MobileNetV2 has been evaluated on ImageNet for classification and adapted for VOC and COCO for segmentation, achieving an inference time of about 0.03 seconds per image on a mobile device for 224x224 inputs. [7]

MobileNetV3: Efficient Neural Network for Mobile Vision Applications MobileNetV3 incorporates advancements such as squeeze-and-excitation modules and the h-swish activation function to improve performance and efficiency. This architecture processes RGB images, typically at 224x224 dimensions, and produces feature maps for segmentation and other vision tasks. MobileNetV3 has been adapted for segmentation datasets like PASCAL VOC, showing an inference time of around 0.02 seconds per image on a mobile device for 224x224 inputs. [8]

4 Image Segmentation

Image segmentation is a process in computer vision and digital image processing, focusing on dividing an image into segments or groups that share similar characteristics, each labelled accordingly [9]. This process operates digitally, utilising the pixel representation of an image to cluster these pixels into meaningful segments effectively. More than merely classifying images, image segmentation takes a step further by not only classifying but also localising various elements within an image. It enhances the classification by delineating the exact boundaries of objects, making it at a more advanced level where it not only identifies but also outlines objects within an image, thereby serving as an advanced form of image classification. Table 11 summarises the conventional methods for image segmentation.

4.1 Types of Segmentation [1]

- **Semantic Segmentation:** This method labels each pixel of an image with a class label, such as "cat," "dog," or "road." All pixels belonging to the same category are treated as a single entity, without distinguishing between individual instances of that category.
- **Instance Segmentation:** This approach extends semantic segmentation by not only classifying each pixel but also distinguishing between separate instances of the same category. For example, in an image with several cats, it identifies and segments each cat individually.
- **Panoptic Segmentation:** Combines semantic and instance segmentation to assign a class label to each pixel (semantic segmentation), while also identifying individual instances of objects (instance segmentation). Provides a comprehensive understanding of the scene, ensuring every pixel is accounted for.

Figure 1: Original Image

Figure 2: Semantic Segmentation Image

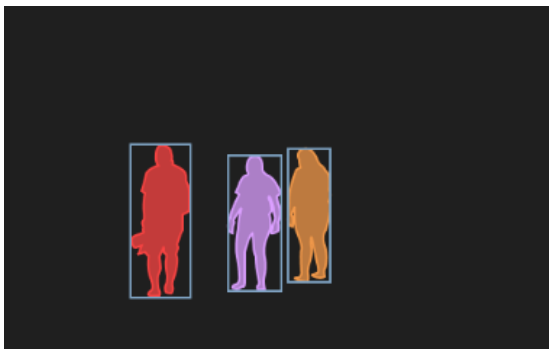

Figure 3: Instance Segmentation Image

Figure 4: Panoptic Segmentation Image

4.2 Semantic Segmentation - Focus of this Project

Semantic segmentation, the focus of this project, involves classifying each pixel in an image into a predefined category. This task is particularly useful for understanding and interpreting complex scenes where multiple objects and regions of interest coexist. By assigning a class label to every pixel, semantic segmentation enables detailed analysis and understanding of the scene.

Table 1: Conventional Methods for Image Segmentation [9]

Method	**Algorithm Description**	**Advantages & Disadvantages**
Thresholding	Divides the image into regions based on intensity values using a chosen threshold.	**Advantages:** Simple to implement, fast computation, effective for high contrast images. **Disadvantages:** Struggles with varying lighting conditions, may not perform well with complex scenes or overlapping intensity values.
Region-Based Segmentation	Separates the objects into different regions based on some threshold value.	**Advantages:** Simple calculations, fast operation speed, performs well with high contrast. **Disadvantages:** Difficulty in accurate segmentation with no significant grayscale difference or overlap of grayscale pixel values.
Edge Detection	Uses discontinuous local features to detect edges and outline objects.	**Advantages:** Effective for images with clear contrast between objects. **Disadvantages:** Not suitable for images with many edges, less effective with low contrast between objects.
Clustering	Divides image pixels into homogeneous clusters based on features like color or intensity.	**Advantages:** Works well on small datasets, generates excellent clusters. **Disadvantages:** High computation time and cost, k-means is distance-based and not suitable for non-convex clusters.

5 Models for Segmentation

This section explores two advanced models tailored for semantic segmentation: DeepLabv3+ and SegFormer. DeepLabv3+ enhances the DeepLabv3 architecture with advanced atrous convolution and a refined decoder module, achieving state-of-the-art performance in capturing detailed semantic information. On the other hand, SegFormer leverages a hierarchical vision transformer backbone and efficient token-based processing to excel in both efficiency and accuracy across diverse datasets.

5.1 DeepLabV3+ Model

DeepLabv3+ is an advanced model for semantic image segmentation, which builds upon the DeepLabv3 architecture [5]. Semantic segmentation involves labeling each pixel in an image with a class from a predefined set of categories. DeepLabv3+ achieves state-of-the-art performance by combining atrous convolution and a novel decoder module to refine the segmentation results.

5.1.1 Architecture

The architecture of DeepLabv3+ consists of two main components: the encoder and the decoder [5].

Encoder

The encoder is responsible for extracting high-level features from the input image. It uses a backbone network, such as MobileNetV2, MobileNetV3 Large, or MobileNetV3 Small, which is pretrained on a large dataset (e.g., ImageNet) to leverage learned representations. The key features of the encoder include:

- **Atrous Convolution:** Atrous (or dilated) convolution is employed to capture multi-scale context without losing resolution. It allows the network to have a larger field of view without increasing the number of parameters.
- **Atrous Spatial Pyramid Pooling (ASPP):** ASPP is a module that applies atrous convolution with different rates in parallel, effectively capturing information at multiple scales. The outputs of these convolutions are concatenated and further processed to produce the final high-level feature maps.

Decoder

The decoder module is designed to refine the segmentation results by merging the encoder's high-level features with low-level features extracted from the early layers of the backbone network. The main components of the decoder are:

- **Low-Level Feature Processing:** The low-level features from the backbone are first processed through a 1x1 convolution and batch normalization to reduce the number of channels.
- **Feature Upsampling and Concatenation:** The high-level features from the ASPP module are upsampled and concatenated with the processed low-level features. This combination helps in recovering spatial details lost during downsampling in the encoder.
- **Final Segmentation:** The concatenated features are passed through a few convolutional layers to produce the final segmentation map.

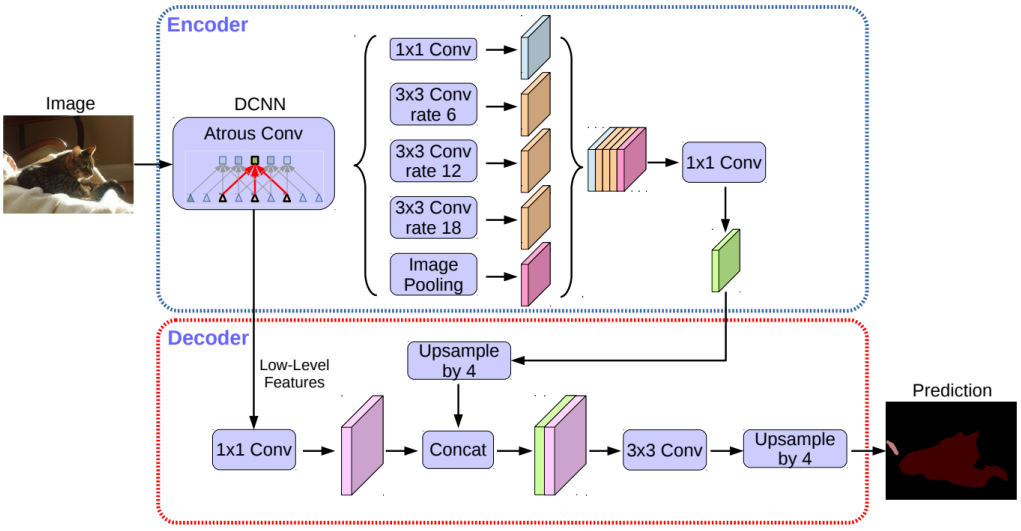

Figure 5: DeepLabv3+ Architecture

5.2 SegFormer Model

SegFormer is a versatile transformer-based model designed for efficient and high-quality semantic segmentation tasks [6]. The architecture is built around a hierarchical vision transformer backbone and is noted for its efficient design and strong performance across various datasets, including ADE20K and Cityscapes.

5.2.1 Architecture

The architecture of SegFormer consists of two main components: the hierarchical transformer encoder and the efficient token-based processing mechanism [6].

Hierarchical Transformer Encoder

The hierarchical transformer encoder is responsible for capturing multi-scale features from the input image. This encoder processes the image at different levels of detail, which is essential for accurate segmentation tasks. The key features of the encoder include:

- **Efficient Attention Mechanism:** The spatial reduction attention mechanism reduces computational complexity by decreasing the number of tokens involved in the attention calculation. This ensures efficient processing without losing the structural information necessary for dense pixel tasks like semantic segmentation.
- **Multi-Scale Feature Extraction:** The encoder captures features at multiple scales, allowing the model to handle various levels of detail effectively. This is achieved through a combination of different transformer layers that operate on various resolutions of the input image.

Token Merging Strategy

The token merging strategy in SegFormer is designed to improve computational efficiency while preserving important spatial information. The main components of this strategy are:

- **Adaptive Token Merging:** Similar tokens are merged to reduce the number of tokens processed. This strategy ensures that small objects are not lost and that important spatial details are preserved.
- **SegFormer++ Enhancements:** The SegFormer++ extension further optimizes the token merging strategy and the attention mechanism. It includes adaptive token merging rates tailored for different stages of the processing pipeline, providing a balance between speed and accuracy.

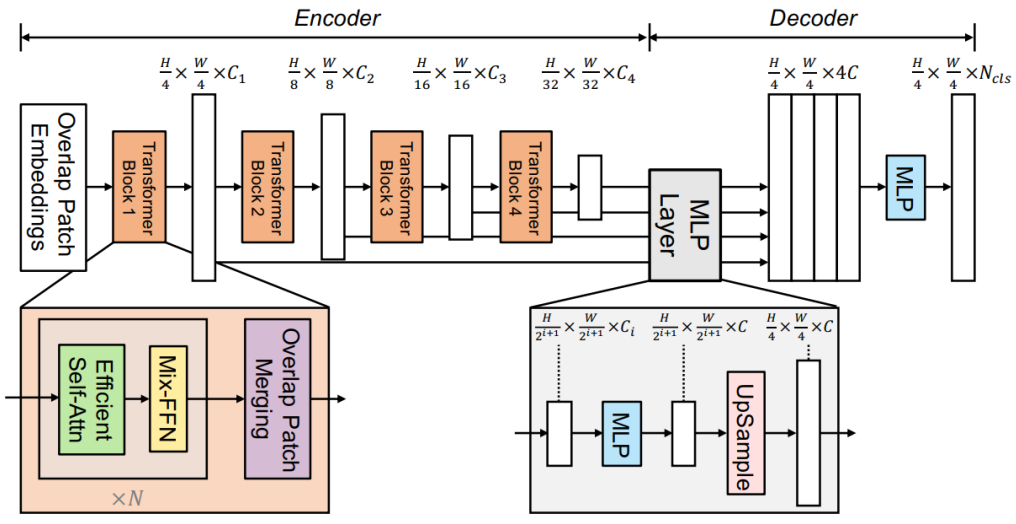

Figure 6: Segformer Architecture

5.3 Model Used in the Project

We have used the DeepLabv3+ model for this project due to its advanced capabilities in semantic segmentation. DeepLabv3+ enhances the original DeepLabv3 architecture with atrous convolution and a novel decoder module, significantly improving the accuracy and detail of segmentation results [5]. The flexibility to integrate

efficient backbones like MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small allows us to balance performance and computational efficiency, making DeepLabv3+ an ideal choice for our segmentation tasks on the PASCAL VOC 2012 and ADE20K datasets.

6 MobileNet Backbones

MobileNet backbones, including MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small, represent a series of innovative neural network architectures tailored for mobile and resource-constrained environments. Developed by Google, these models integrate advanced techniques such as depthwise separable convolutions, inverted residuals, and efficient activation functions to balance computational efficiency with high performance. Each variant, from the efficient MobileNetV2 to the enhanced MobileNetV3 Large and resource-friendly MobileNetV3 Small, offers distinct advantages suited for real-time applications on devices with limited computational resources.

6.1 MobileNetV2 Model

MobileNetV2, developed by Google, represents a significant advancement in the field of efficient neural networks designed specifically for mobile and resource-constrained environments [7]. Building upon the success of its predecessor, MobileNetV1, MobileNetV2 introduces a novel architecture that balances computational efficiency and accuracy. The key innovation in MobileNetV2 is the introduction of inverted residual structures combined with linear bottlenecks. This architectural design allows for the reduction of memory and computational cost by factorizing standard convolutions into depthwise separable convolutions, which consist of depthwise and pointwise convolutions. The inverted residual structure, where shortcut connections are added between thin bottleneck layers, enhances feature reuse and minimizes the number of parameters needed. Moreover, linear bottlenecks, which use linear activation functions instead of non-linear ReLU activations in the narrow layers, help preserve critical information that might otherwise be lost. These enhancements make MobileNetV2 an ideal choice for real-time applications on devices with limited computational power, providing a powerful yet efficient solution for a wide range of computer vision tasks.

6.1.1 Architecture Details

The architecture of MobileNetV2 mainly consists of: depthwise separable convolutions, inverted residuals and linear bottlenecks [7].

Inverted Residuals and Linear Bottlenecks

- **Inverted Residuals:** Traditional residual blocks used in networks like ResNet add residuals (shortcut connections) directly between input and output feature maps. In contrast, MobileNetV2 uses inverted residuals where the shortcut connections are added between narrow bottleneck layers rather than wide layers. This means the input to a residual block is first expanded (using a pointwise convolution) before applying depthwise separable convolutions, and then compressed back to a narrow bottleneck layer. This structure allows the network to capture more complex features with fewer parameters.
- **Linear Bottlenecks:** Unlike traditional convolutional layers that use ReLU activations which can lose information, the bottleneck layers in MobileNetV2 use a linear activation function. This helps to preserve more information, especially when the number of channels is reduced drastically, ensuring that the essential features are not lost.

Depthwise Separable Convolutions

- **Depthwise Convolutions:** Instead of applying a standard convolution to the input, which convolves each filter with all input channels, depthwise convolution applies a single convolutional filter per input channel. This greatly reduces the computational cost.
- **Pointwise Convolutions:** Following the depthwise convolution, a 1x1 pointwise convolution is applied to combine the outputs of the depthwise convolution. This step is responsible for changing the dimensionality of the feature space.
- **Efficiency:** By breaking down standard convolutions into depthwise and pointwise convolutions, MobileNetV2 achieves significant reductions in computational complexity and the number of parameters, making it suitable for mobile and embedded applications.

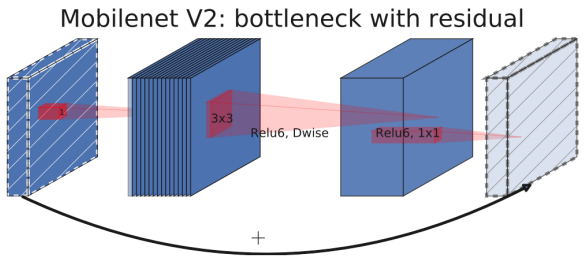

Figure 7: MobileNetV2 layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck rather than expansion).

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	-	

Figure 8: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size.

6.2 MobileNetV3 Large Model

MobileNetV3 Large, developed by Google, represents a significant evolution in the MobileNet family of architectures designed for mobile and resource-constrained environments [8]. It combines the strengths of MobileNetV2 and introduces new architectural innovations to further enhance performance and efficiency. Key innovations in MobileNetV3 include the use of the new h-swish activation function, the introduction of squeeze-and-excitation (SE) modules, and a novel network design search. These enhancements allow MobileNetV3 Large to achieve a better balance between accuracy and computational cost, making it highly suitable for real-time applications on devices with limited computational power.

6.2.1 Architecture Details

The architecture of MobileNetV3 Large is described as follows:

h-swish Activation Function and SE Modules [8]

- **h-swish Activation Function:** MobileNetV3 Large replaces the traditional ReLU activation with the h-swish activation function, defined as:

$$\text{h-swish}(x) = x \cdot \frac{\text{ReLU6}(x+3)}{6}$$

  where $\text{ReLU6}(x) = \max(0, \min(x, 6))$. This activation function improves the expressiveness of the network while maintaining computational efficiency.

- **SE Modules:** Squeeze-and-Excitation (SE) modules are integrated into the architecture to adaptively recalibrate feature maps. The SE module performs channel-wise feature recalibration by modeling interdependencies between channels, enhancing the network's representational power.

Depthwise Separable Convolutions [8]

- **Depthwise Convolutions:** These convolutions apply a single convolutional filter per input channel, significantly reducing the computational cost compared to standard convolutions.

- **Pointwise Convolutions:** Following the depthwise convolution, a 1×1 pointwise convolution is applied to combine the outputs of the depthwise convolution, adjusting the dimensionality of the feature space.

- **Efficiency:** By decomposing standard convolutions into depthwise and pointwise convolutions, MobileNetV3 Large achieves significant reductions in computational complexity and the number of parameters, enhancing its suitability for mobile and embedded applications.

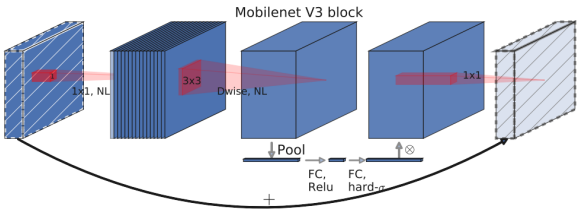

Figure 9: MobileNetV3 Architecture (MobileNetV2 + Squeeze-and-Excite)

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Figure 10: Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. s denotes stride.

6.3 MobileNetV3 Small Model

MobileNetV3 Small, also developed by Google, is another variant in the MobileNetV3 family that prioritizes efficiency and low computational cost, making it even more suitable for extremely resource-constrained environments [8]. Like its larger counterpart, MobileNetV3 Small incorporates several key innovations such as the h-swish activation function, SE modules, and an optimized architecture obtained through network design search. These enhancements allow MobileNetV3 Small to provide a highly efficient solution for various computer vision tasks, including real-time semantic segmentation on devices with very limited computational power.

6.3.1 Architecture Details

The architecture of MobileNetV3 Small is described as follows:

h-swish Activation Function and SE Modules [8]

- **h-swish Activation Function:** MobileNetV3 Small uses the h-swish activation function to improve the expressiveness of the network while maintaining computational efficiency.
- **SE Modules:** Squeeze-and-Excitation (SE) modules are integrated into the architecture to adaptively recalibrate feature maps, enhancing the network's representational power by modeling interdependencies between channels.

Depthwise Separable Convolutions [8]

- **Depthwise Convolutions:** These convolutions apply a single convolutional filter per input channel, significantly reducing the computational cost compared to standard convolutions.
- **Pointwise Convolutions:** Following the depthwise convolution, a 1×1 pointwise convolution is applied to combine the outputs of the depthwise convolution, adjusting the dimensionality of the feature space.
- **Efficiency:** By decomposing standard convolutions into depthwise and pointwise convolutions, MobileNetV3 Small achieves significant reductions in computational complexity and the number of parameters, enhancing its suitability for mobile and embedded applications.

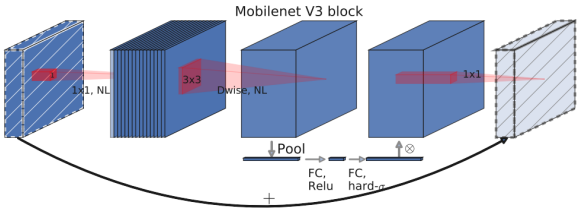

Figure 11: MobileNetV3 Architecture (MobileNetV2 + Squeeze-and-Excite)

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Figure 12: Specification for MobileNetV3-Small. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes the type of nonlinearity used. Here, HS denotes h-swish and RE denotes ReLU. NBN denotes no batch normalization. s denotes stride.

7 Dataset Description

7.1 PASCAL VOC 2012 Dataset

The PASCAL VOC (Visual Object Classes) 2012 dataset is a widely used benchmark in the field of computer vision, specifically for object classification, detection, and semantic segmentation tasks [10][11]. Key features of the VOC 2012 dataset include:

- **Categories:** It consists of 20 object categories plus one background category, totaling 21 classes. These classes cover a diverse range of everyday objects and animals.
- **Image Annotations:** Each image in the dataset is annotated with pixel-level segmentation masks, where every pixel is labeled with a class from the predefined set. This facilitates fine-grained evaluation and training of semantic segmentation models.
- **Image Variability:** The dataset includes a diverse set of images captured in various environments, lighting conditions, and perspectives, enhancing the robustness and generalizability of trained models.
- **Training and Validation Sets:** The VOC 2012 dataset is split into training and validation subsets. The training set contains a significant number of images for model training, while the validation set is used to evaluate model performance on unseen data.

7.2 ADE20K Dataset

The ADE20K (MIT Scene Parsing Benchmark) dataset is designed for scene parsing, which involves understanding the entire scene in an image and assigning a semantic label to each pixel [10][12][13]. Key characteristics of the ADE20K dataset include:

- **Semantic Categories:** ADE20K consists of over 150 semantic categories, covering a wide range of scenes and objects. This extensive label set enables detailed scene understanding and segmentation.

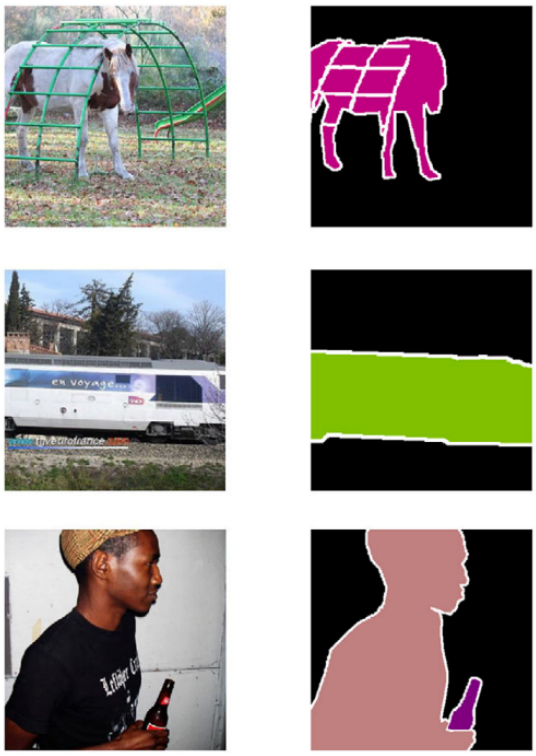

Figure 13: Pascal VOC 2012 Dataset Examples.(left is the image and right is the ground truth)

Figure 14: ADE20K Dataset Examples.(left is the image and right is the ground truth)

- **Rich Annotations:** Each image in the ADE20K dataset is densely annotated with pixel-level labels, providing comprehensive ground truth for training and evaluation. This level of annotation supports fine-grained segmentation tasks.
- **Complex Scenes:** Images in ADE20K capture complex scenes with multiple objects, occlusions, and varying spatial arrangements. This complexity challenges models to learn robust features and contextual understanding.
- **Dataset Size:** ADE20K contains approximately 20,000 images for training and 2,000 images for validation, ensuring a substantial dataset size suitable for training deep learning models effectively.

8 Implementation Methodology

8.1 Encoder

The encoder is essential for feature extraction from input images, leveraging pretrained backbone networks. We employed three different backbones: MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small. Each backbone provides unique advantages in terms of accuracy and efficiency, making them suitable for various segmentation tasks.

8.1.1 MobileNetV2

- **Low-level features:** Extracted from the first four layers (`features[:4]`) with 24 channels, capturing fine-grained details such as edges and textures.
- **High-level features:** Extracted from the remaining layers up to the penultimate layer (`features[4:-1]`) with 320 channels, capturing more abstract features necessary for understanding the overall context of the image.

8.1.2 MobileNetV3 Large

- **Low-level features:** Extracted from the first seven layers (`features[:7]`) with 40 channels, capturing essential fine-grained details.

- **High-level features:** Extracted from the subsequent layers (`features[7:]`) with 960 channels, capturing abstract, high-level features for contextual understanding.

8.1.3 MobileNetV3 Small

- **Low-level features:** Extracted from the first four layers (`features[:4]`) with 24 channels, similar to MobileNetV2.
- **High-level features:** Extracted from the remaining layers (`features[4:]`) with 576 channels, capturing high-level contextual information.

The extraction of these features was facilitated using the `IntermediateLayerGetter` class, which efficiently retrieves features from specific intermediate layers of each backbone network.

8.2 ASPP (Atrous Spatial Pyramid Pooling) Module

The ASPP module captures multi-scale contextual information through atrous convolutions at different dilation rates, enabling the model to incorporate information from various spatial scales and enhancing its ability to segment objects of different sizes.

8.2.1 Implementation

- **Parallel branches:** Several parallel branches of atrous convolutions with different dilation rates (e.g., 6, 12, 18 for an output stride of 16) to capture context at multiple scales.
- **Global average pooling branch:** Ensures holistic information gathering.
- **Batch normalization and activation layers:** Follow each convolutional operation to stabilize and activate the features appropriately.

8.3 Decoder

The decoder module refines segmentation outputs by combining low-level and high-level features to produce detailed segmentation maps. It reconstructs the spatial resolution of features and integrates context-rich representations from the encoder.

8.3.1 Functionality

- **1x1 convolution:** Adjusts the dimensions of low-level features.
- **Bilinear interpolation:** Aligns high-level features with the spatial dimensions of low-level features.
- **Concatenation:** Combines the adjusted high-level and low-level features to fuse multi-level contextual information.
- **Convolutional layers:** Generate the final pixel-wise predictions, ensuring accurate and detailed segmentation maps.

8.4 Segmentation Model Integration

The overall segmentation model integrates the encoder and decoder components to form a unified architecture capable of end-to-end semantic segmentation.

8.4.1 MobileNetV2 Integration

- **Loading and Modifying MobileNetV2:** The model is loaded using PyTorch's `torchvision` library.
- **Layer Selection:**
  - **Low-level features:** Extracted from `features[:4]`.
  - **High-level features:** Extracted from `features[4:-1]`.
- `Intermediate Layer Getter:` Facilitates extraction of specific layers.
- **ASPP Module:** Configured with dilation rates of 6, 12, and 18.

- **Decoder:**
  - Low-level features processed through 1x1 convolution to 48 channels.
  - High-level features upsampled to match low-level feature dimensions.
  - Concatenated and processed through convolutional layers to produce final segmentation map.

8.4.2 MobileNetV3 Large Integration

- **Loading and Modifying MobileNetV3 Large:** Loaded using PyTorch's `torchvision` library.
- **Layer Selection:**
  - **Low-level features:** Extracted from `features[:7]`.
  - **High-level features:** Extracted from `features[7:]`.
- `Intermediate Layer Getter:` Facilitates extraction of specific layers.
- **ASPP Module:** Configured with dilation rates of 6, 12, and 18.
- **Decoder:**
  - Low-level features processed through 1x1 convolution to 48 channels.
  - High-level features upsampled to match low-level feature dimensions.
  - Concatenated and processed through convolutional layers to produce final segmentation map.

8.4.3 MobileNetV3 Small Integration

- **Loading and Modifying MobileNetV3 Small:** Loaded using PyTorch's `torchvision` library.
- **Layer Selection:**
  - **Low-level features:** Extracted from `features[:4]`.
  - **High-level features:** Extracted from `features[4:]`.
- `Intermediate Layer Getter:` Facilitates extraction of specific layers.
- **ASPP Module:** Configured with dilation rates of 6, 12, and 18.
- **Decoder:**
  - Low-level features processed through 1x1 convolution to 48 channels.
  - High-level features upsampled to match low-level feature dimensions.
  - Concatenated and processed through convolutional layers to produce final segmentation map.

In all cases, the encoder extracts feature maps from input images using the specified backbone network (MobileNetV2, MobileNetV3 Large, or MobileNetV3 Small). These features undergo ASPP processing to capture comprehensive spatial context. The decoder then refines these features by combining multi-level representations and generating detailed segmentation predictions.

9 Parameters and Hyperparameters

The performance of deep learning models, particularly for tasks such as semantic segmentation, is significantly influenced by the chosen parameters and hyperparameters. In this study, we trained DeepLabv3+ models with different MobileNet backbones on the PASCAL VOC 2012 and ADE20K datasets. Tables 2 and 3 outline the specific parameters and hyperparameters utilized for these experiments.

For the PASCAL VOC 2012 dataset, the input image size was fixed at 224x224 pixels, and the number of classes was set to 21. The backbone architectures included MobileNetV2 [7], MobileNetV3 Large [8], and MobileNetV3 Small [8], with the DeepLabv3+ decoder and an output stride of 16. The hyperparameters, such as the learning rate, optimizer, batch size, number of epochs, momentum, weight decay, and loss function, were carefully selected to optimize performance.

Similarly, for the ADE20K dataset, the input image size remained at 224x224 pixels, but the number of classes increased to 151. The same backbone architectures and decoder were used, with a slightly different set of hyperparameters tailored for the dataset's complexity. These configurations are detailed in Table 3, highlighting the adjustments made to account for the diverse and extensive nature of the ADE20K dataset.

Table 2: Parameters and Hyperparameters for DeepLabv3+ with Different Backbones for PASCAL VOC 2012 Dataset

Category	**Arguments**	**DeepLabv3+ and MobileNetV2**	**DeepLabv3+ and MobileNetV3 Large**	**DeepLabv3+ and MobileNetV3 Small**
Fixed Parameters	Input Image Size	224x224	224x224	224x224
	Number of Classes	21	21	21
Architecture	Backbone	MobileNetV2	MobileNetV3 Large	MobileNetV3 Small
	Decoder	DeepLabv3+	DeepLabv3+	DeepLabv3+
	Output Stride	16	16	16
Hyperparameters	Learning Rate	0.001	0.001	0.001
	Optimizer	SGD	SGD	SGD
	Batch Size	10	10	10
	Epochs	50	50	50
	Momentum	0.9	0.9	0.9
	Weight Decay	1e-4	1e-4	1e-4
	Loss Function	Cross Entropy	Cross Entropy	Cross Entropy

Table 3: Parameters and Hyperparameters for DeepLabv3+ with Different Backbones for ADE20K Dataset

Category	**Arguments**	**DeepLabv3+ and MobileNetV2**	**DeepLabv3+ and MobileNetV3 Large**	**DeepLabv3+ and MobileNetV3 Small**
Fixed Parameters	Input Image Size	224x224	224x224	224x224
	Number of Classes	151	151	151
Architecture	Backbone	MobileNetV2	MobileNetV3 Large	MobileNetV3 Small
	Decoder	DeepLabv3+	DeepLabv3+	DeepLabv3+
	Output Stride	16	16	16
Hyperparameters	Learning Rate	0.0005	0.0005	0.0005
	Decay Rate Factor	0.1	0.1	0.1
	Learning Rate Step Size	2	2	2
	Optimizer	SGD	SGD	SGD
	Batch Size	6	6	6
	Epochs	50	50	50
	Momentum	0.9	0.9	0.9
	Weight Decay	0.0002	0.0002	0.0002
	Loss Function	Cross Entropy	Cross Entropy	Cross Entropy

10 Evaluation Metrics

The effectiveness of semantic segmentation models is assessed through a range of evaluation metrics that provide insights into their accuracy, efficiency, and resource requirements. These metrics help in understanding the trade-offs between model performance and computational costs, especially crucial for deploying models in real-world applications. The below are some metrics used for evaluation.

1. **Inference Time:** Inference Time [14] refers to the average time taken by the model to process and generate predictions for one image. It is typically measured in seconds and directly impacts the model's usability in real-time applications.

2. **GPU Memory Usage:** GPU Memory Usage [15] indicates the amount of memory (in megabytes, MB) consumed by the model during training or inference on a GPU. It reflects the model's memory requirements and scalability, impacting the choice of hardware for deployment.

3. **Mean Intersection over Union (mIoU):** Mean Intersection over Union (mIoU) [5] evaluates the quality of the segmentation masks produced by the model. It calculates the average Intersection over Union (IoU) score across all classes, where IoU measures the overlap between predicted and ground truth masks for each class.

$$\text{IoU} = \frac{\text{True Positive (TP)}}{\text{True Positive (TP)} + \text{False Positive (FP)} + \text{False Negative (FN)}}$$

$$\text{mIoU} = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i}$$

   Where:

   - TP_i: True Positives for class i
   - FP_i: False Positives for class i
   - FN_i: False Negatives for class i
   - N: Number of classes

4. **Pixel Accuracy (PA):** Pixel Accuracy (PA) [16] measures the percentage of correctly classified pixels in the segmentation output compared to the ground truth. It provides a pixel-level assessment of segmentation accuracy across all classes.

$$\text{Pixel Accuracy} = \frac{\text{Number of correctly classified pixels}}{\text{Total number of pixels}}$$

5. **Computational Complexity:** Computational complexity [17] refers to the amount of computational resources (typically measured in FLOPs - Floating Point Operations) required to perform inference or training with the model. It indicates the model's efficiency in terms of processing power.

6. **Number of Parameters:** The number of parameters [7] represents the total count of learnable weights in the model, including both trainable and non-trainable parameters. It provides insights into the model's capacity and potential for overfitting, as well as its memory requirements.

7. **Average Power Consumption:** Average power consumption [18] during training and evaluation measures the average electrical power consumed by the GPU or system. It is typically measured in watts (W) and helps assess the energy efficiency and cost of running the model over time.

11 Results

Table 4 and 5 provide a detailed comparison of the performance metrics for the DeepLabv3+ model with different MobileNet backbones, namely MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small, on both the PASCAL VOC 2012 and ADE20K datasets. These comparisons include accuracy, inference time, GPU memory usage, mean Intersection over Union (mIoU), pixel accuracy, computational complexity, number of parameters, training time, and power consumption during training and evaluation.

The performance on the PASCAL VOC 2012 dataset (Table 4) indicates that MobileNetV3 Small achieves the fastest inference time of 9.11 ms and the lowest GPU memory usage of 1135.6835 MB. All models exhibit similar accuracy and mIoU, with MobileNetV2 showing slightly higher mIoU at 47.66%. On the ADE20K dataset (Table 5), the models also show comparable performance, though the pixel accuracy and mIoU values are generally lower due to the increased complexity of this dataset. MobileNetV2, again, slightly outperforms the others in terms of mIoU at 0.0976981485.

Table 4: Performance Comparison of Semantic Segmentation Models on PASCAL VOC 2012 Dataset

Metric	**DeepLabv3+ and MobileNetV2**	**DeepLabv3+ and MobileNetV3 Large**	**DeepLabv3+ and MobileNetV3 Small**
Inference Time (ms)	9.06	10.24	9.11
Final GPU Memory Usage (MB)	1136.0395	1135.5429	1135.6835
mIoU	47.66%	47.51%	47.60%
Pixel Accuracy	94.38%	94.40%	94.39%
Computational Complexity (GMac/MMac)	2.65 GMac	1.14 GMac	833.53 MMac
Number of Parameters (M)	5.23	11.4	6.24
Training Time (seconds)	927.77	853.99	765.48
Average Power Consumption during Training (W)	40.74	62.56	53.71
Average Power Consumption during Evaluation (W)	41.74	37.26	34.09

Table 5: Performance Comparison of Semantic Segmentation Models on ADE20K Dataset

Metric	**DeepLabv3+ and MobileNetV2**	**DeepLabv3+ and MobileNetV3 Large**	**DeepLabv3+ and MobileNetV3 Small**
Inference Time	0.0033428669	0.0038104057	0.0034222603
mIoU	0.0976981485	0.0969983200	0.0769910713
Pixel Accuracy	65.2671295676	64.8039823822	60.8749014340
Computational Complexity (GMac/MMac)	2.65 GMac	1.14 GMac	833.53 MMac
Number of Parameters (M)	5.23	11.4	6.24

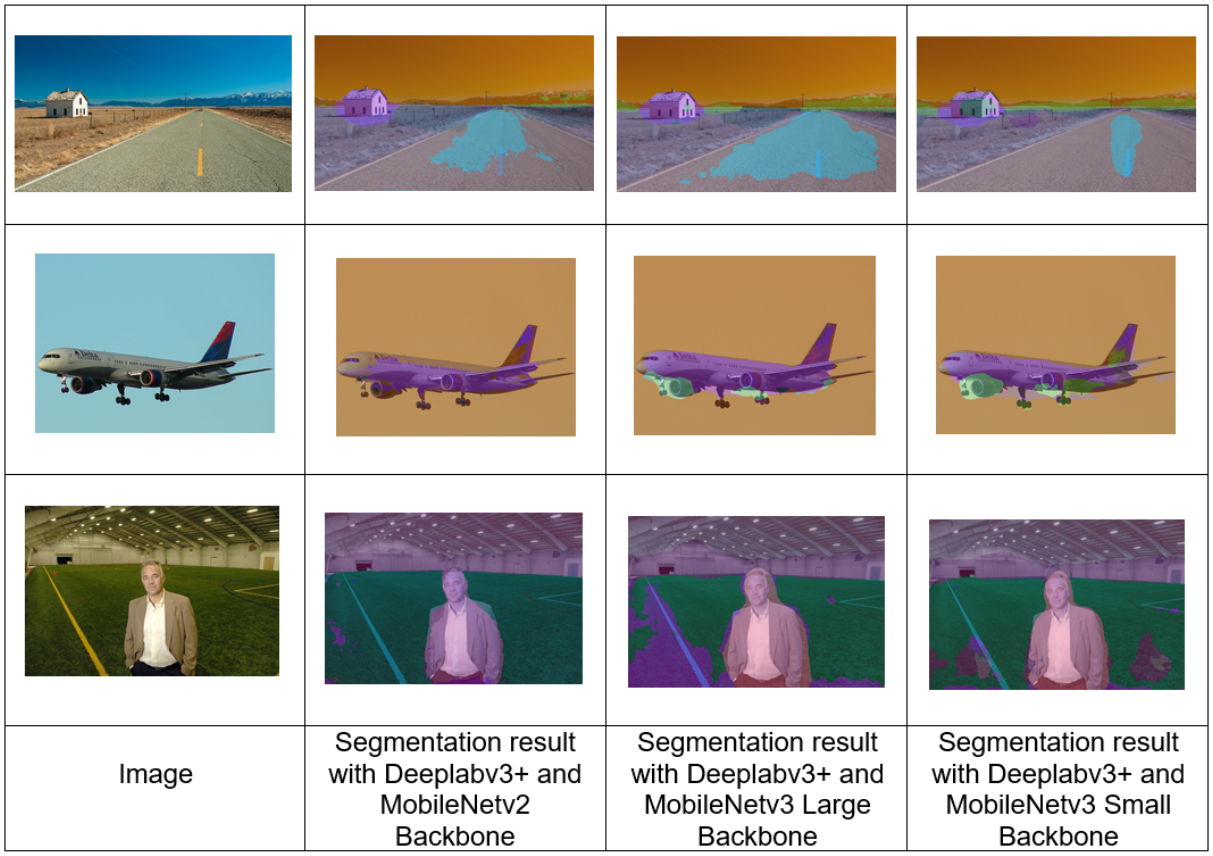

Figure 15: Some examples from the above models

12 Discussion

The results demonstrate the trade-offs between model complexity, computational efficiency, and segmentation accuracy when using different MobileNet backbones with the DeepLabv3+ model. The choice of backbone significantly impacts key performance metrics.

MobileNetV3 Small offers the fastest inference time and lowest GPU memory usage, making it well-suited for resource-constrained environments. Despite its lower computational complexity and fewer parameters, it maintains comparable accuracy and mIoU to the other models, indicating its efficiency and effectiveness in real-time applications.

MobileNetV2 shows a balance between computational complexity and accuracy, with the highest mIoU on both datasets. This makes it a suitable choice for applications requiring moderate computational resources while maintaining high segmentation quality.

MobileNetV3 Large, despite its higher computational requirements, provides slightly higher accuracy and pixel accuracy, making it preferable for scenarios where computational resources are less constrained and accuracy is paramount.

Overall, these insights guide the selection of appropriate models based on the specific requirements and constraints of various applications. The balance between inference time, memory usage, and segmentation accuracy must be carefully considered to optimize the deployment of semantic segmentation models in real-world scenarios. Given that the project requires less inference time and better Mean Intersection over Union (MIoU), it is recommended to choose MobileNetV2 as the backbone for DeepLabv3+ based on the results it has provided.

13 Conclusions

In this study, we evaluated various state-of-the-art models for semantic segmentation, including DeepLabv3+, SegFormer, and MobileNet variants, specifically focusing on MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small backbones. Each model has unique strengths in terms of architecture, computational efficiency, and performance on benchmark datasets. DeepLabv3+ stands out for its ability to capture multi-scale context and produce detailed segmentation results. SegFormer offers a novel transformer-based approach that combines efficiency and accuracy. MobileNetV2, MobileNetV3 Large, and MobileNetV3 Small are lightweight models designed for mobile and resource-constrained environments, balancing performance with lower computational requirements. Our evaluations showed that all MobileNet variants provided efficient performance, with MobileNetV3 Small and MobileNetV2 achieving an average inference time of 0.0086 seconds per image and MobileNetV3 Large achieving 0.0102 seconds per image, all with minimal GPU memory usage. These insights help in selecting the right model for specific deployment scenarios, considering the trade-offs between complexity, accuracy, and efficiency. Given that the project requires less inference time and better Mean Intersection over Union (MIoU), it is recommended to choose MobileNetV2 as the backbone for DeepLabv3+ based on the results it has provided.

References

[1] Image segmentation guide. https://www.v7labs.com/blog/image-segmentation-guide.

[2] PyTorch. Quantization documentation. https://pytorch.org/docs/stable/quantization.html, 2020.

[3] Libraria AI. Computer vision techniques in ai: Changing the way machines see. URL https://libraria.ai/blog/computer-vision-techniques-in-ai-changing-the-way-machines-see/.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[5] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[6] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *Advances in Neural Information Processing Systems*, volume 34, pages 12077–12090, 2021.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[8] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[9] Introduction to image segmentation techniques in python. https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/.

[10] Bowen Cheng, Alexander G Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *Advances in Neural Information Processing Systems*, volume 34, pages 17864–17875, 2021.

[11] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, and Philip HS Torr. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6881–6890, 2021.

[12] Zhengyuan Chen, Wei Wang, Zhiding Wang, Dingcheng Chen, Yu Qiao, Youngtae Ko, Tal Hassner, and Stephen Tyree. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. *arXiv preprint arXiv:2106.13711*, 2021.

[13] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. *arXiv preprint arXiv:2105.05633*, 2021.

[14] Rudra PK Poudel, Stefan Liwicki, and Roberto Cipolla. Fast-scnn: Fast semantic segmentation network. In *British Machine Vision Conference (BMVC)*, 2019.

[15] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *European Conference on Computer Vision (ECCV)*, pages 552–568. Springer, 2018.

[16] Changqian Yu, Jingbo Wang, Chang Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *European Conference on Computer Vision (ECCV)*, pages 325–341. Springer, 2018.

[17] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 6848–6856, 2018.

[18] Chaojian Xiao, Wei Zhang, Yujie Liu, and Athanasios V Vasilakos. Energy-efficient neural networks for object detection on mobile devices. In *IEEE Transactions on Neural Networks and Learning Systems*, volume 30, pages 1963–1975. IEEE, 2019.

[19] K. O'shea and R. Nash. An introduction to convolutional neural networks. 2015.

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

A Appendix A: Personal Learning and Development Journey

My initial learning involved understanding the fundamental architecture of CNNs, including concepts like convolutional layers, pooling layers, and fully connected layers. I started by watching a lot of YouTube tutorials, reading research papers, and taking online courses [19]. These resources helped me build a strong understanding of how CNNs work and how they are used in tasks like image processing.

A.1 Building a Custom CNN

My initial attempts at building a custom CNN focused on the CIFAR-100 dataset, a challenging dataset due to its complexity and diversity. The experience was both rewarding and instructive, as I encountered and overcame various challenges, such as tuning hyperparameters, managing overfitting, and debugging issues related to vanishing gradients. These efforts laid a strong foundation for understanding CNN architectures. I further expanded my experiments by applying custom CNNs to other datasets like CIFAR-10 and Fashion MNIST, which provided deeper insights into model generalization across different types of data.

A.2 Experimenting with Backbone Models

To gain a deeper understanding of different architectural approaches, I experimented with various backbone models:

- **ResNet18 Backbone [20]**: This provided an introduction to residual learning, allowing me to understand how skip connections help in training deeper networks.
- **MobileNetV2 Backbone [7]**: Its efficient architecture using inverted residuals and linear bottlenecks demonstrated a balance between complexity and performance.
- **MobileNetV3 Large Backbone [8]**: Integrating h-swish activation and squeeze-and-excitation modules showed significant improvements in performance, especially for more complex scenes.
- **MobileNetV3 Small Backbone [8]**: Despite its compact size, this model delivered impressive results, emphasizing the advancements in efficient network design for mobile applications.

These experiments informed my decisions for model selection in practical applications and enhanced my understanding of how different backbone models impact the performance and efficiency of CNNs.

A.3 Final Code and Implementation

The culmination of my learning and experimentation involved integrating these insights into building custom DeepLabv3+ models with different MobileNet backbones on the VOC 2012 dataset. This process included selecting the most suitable backbone models based on prior experiments and integrating them with the DeepLabv3+ architecture. I optimized the training process by refining learning rate schedules, incorporating advanced data augmentation techniques, and utilizing mixed precision training to accelerate computation without sacrificing accuracy. The final phase involved implementing these models on the ADE20K dataset, testing their performance, and drawing conclusions from the results.

A.4 Conclusion

Reflecting on this journey, I see significant growth from my initial understanding of CNNs to the successful implementation of state-of-the-art segmentation models. Each challenge encountered and overcome has enriched my knowledge and skills. This journey has solidified my passion for computer vision and continues to inspire my pursuit of innovation in this field. To see my works, please follow this link and this link.

B Appendix B: Quantization using PyTorch [2]

Quantization is a technique used to reduce the model size and inference time by converting the weights and activations of a neural network from floating-point (32-bit) to a lower precision, such as 8-bit integers. PyTorch provides robust support for quantization, which can be broadly categorized into Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT).

B.1 Post-Training Quantization (PTQ)

Post-Training Quantization is applied to a pre-trained model without requiring additional training. PTQ can be further divided into static and dynamic quantization:

B.1.1 Static Quantization

In static quantization, both weights and activations are quantized. This involves calibrating the activations by running a subset of the training data through the model to determine the optimal scaling factors. The model is first prepared for static quantization by fusing certain layers to improve performance, specifying a quantization configuration, and then calibrating the model using a representative dataset. Once calibrated, the model is converted to a quantized version that can be used for inference with reduced computational resources.

B.1.2 Dynamic Quantization

Dynamic quantization only quantizes the weights, while the activations are dynamically quantized during inference. This method is simpler and generally faster for models that are dominated by matrix multiplications, such as RNNs and Transformers. To apply dynamic quantization, the model is first prepared, and then specific layers, such as linear layers, are quantized. Despite the simplicity and potential speed advantages, our experiments with CNNs showed that dynamic quantization did not yield significant performance improvements.

B.2 Quantization-Aware Training (QAT)

Quantization-Aware Training incorporates quantization into the training process itself, simulating the effects of quantization during both forward and backward passes. This approach often results in better accuracy and performance compared to PTQ because the model is trained to be robust to the quantization noise. The model is prepared for QAT by fusing layers and specifying a quantization configuration. Training proceeds as usual, but with the quantization effects simulated during training. Once training is complete, the model is converted to a quantized version that typically offers improved accuracy and efficiency for inference.

B.3 Future Work

As PyTorch continues to advance its quantization techniques, future work can explore Quantization-Aware Training (QAT) more deeply. QAT holds significant potential for reducing resource utilization and improving inference time while maintaining accuracy. Further research into optimizing quantization for various neural network architectures, including CNNs, can lead to broader adoption and better performance in real-world applications.