Advanced Artificial Intelligence

TAFERE GEDION LEUL[1, a]

[1]Xi'an Technological University, Xi'an, Shaanxi, China

[a]gedionleul@gmail.com

Abstract. This report details the development of a semantic segmentation model using the DeepLabV3+ architecture with a Mobile Net backbone, trained on the Pascal VOC2012 dataset. The objective was to create a computationally efficient model capable of generating accurate pixel-wise segmentation masks (single-channel grayscale) for real-world applications. The project encompassed data preprocessing, model construction, training (utilizing Tensor Board for visualization), and inference. Key learning outcomes included managing segmentation data, designing/tuning neural networks, and optimizing training processes.

Keywords: Semantic Segmentation; DeepLabV3+; Mobile Net; Pascal VOC2012

1. Introduction

Semantic segmentation, a key task in computer vision, involves partitioning images into distinct regions to assign meaningful labels to each pixel. This report documents the development of a semantic segmentation model using the DeepLabV3+ architecture with a Mobile Net backbone, trained and evaluated on the Pascal VOC2012 dataset. The model's lightweight design and effective segmentation capabilities make it well-suited for real-world applications requiring computational efficiency.

In this project, I worked on semantic segmentation using the DeepLabV3+ model with a Mobile Net backbone, leveraging the Pascal VOC2012 dataset. The goal was to develop a model capable of accurately segmenting images by predicting single-channel grayscale masks for each pixel. This involved multiple stages, including data preprocessing, model construction, training, and inference. I chose DeepLabV3+ with Mobile Net because of its balance between performance and efficiency, making it suitable for this task. Throughout the project, I learned to handle segmentation data, design and adjust a neural network, and optimize training processes. Using Tensor Board, I visualized the training and validation progress to better understand the model's performance.

2 Problem Statement

Semantic segmentation is a fundamental task in computer vision, aimed at assigning a semantic label to each pixel in an image. The challenge lies in accurately segmenting objects of varying shapes, sizes, and complexities while maintaining computational efficiency. For this project, the objective is to implement a semantic segmentation model using the Pascal VOC2012 dataset.

The dataset includes labeled images for training and validation, as well as an additional test set without labels for evaluation. The primary goal is to assign a class label to each pixel in an image. This project focuses on performing semantic segmentation on the Pascal VOC2012 dataset, which contains images with 21 classes (20 object classes and 1 background class. And to process and load the dataset effectively, construct a compatible network architecture, select and apply a loss function, and train the model to optimize its performance. The evaluation metric for the project is mean Intersection over Union (mIoU), which measures the accuracy of the model's predictions on the test set.

2.1 Dataset Description. The Pascal VOC2012 dataset is a widely used benchmark in computer vision tasks, particularly for semantic segmentation. It provides annotated images for training and validation, which include pixel-wise labels for 20 object classes and one background class. The dataset is designed to evaluate models on their ability to segment objects of varying sizes, shapes, and complexities in diverse real-world scenes.

https://doi.org/10.61957/joerai-20250206

For this project, the dataset is divided into the following components.

Training Set: Consists of 1,465 labeled images. Includes corresponding segmentation masks.

Validation Set: Comprises 1,450 labeled images with segmentation masks (used for validation). Used for model evaluation during training to monitor and fine-tune performance. Test Set: Includes 400 additional images without annotations. These images are used to evaluate the model's segmentation performance.

The predictions are saved as single-channel grayscale masks. Each pixel in the segmentation masks is labeled with one of the 21 classes, including objects like "person", "car", "dog," and "background".

2.2 Environment specification. This project is developed in Google Collaboratory using the following tools and configurations: Jupyter Notebook: For interactive development and testing. PyTorch: As the primary deep learning framework. Tensor Board: For visualizing training metrics and model performance: Tesla T4 with 16GB of RAM for accelerated computation.

3. Network model structure

The proposed network for semantic segmentation integrates a lightweight and efficient design based on the DeepLabV3+ architecture.

3.1. Input Image. The model accepts input images for semantic segmentation tasks, preprocessed to match the expected input size of the network.

3.2. Backbone: MobileNetV2. MobileNetV2 serves as the feature extractor, leveraging its efficient inverted residual blocks. It captures multi-scale features while maintaining a lightweight structure. The stages of feature extraction progress through increasing channel dimensions, specifically: 24 → 32 → 64 → 96 → 160 → 320 channels.

3.3. Low-Level Features. Early feature maps are extracted to retain fine details, crucial for accurate segmentation boundaries. These are processed through: A 3×3 convolution followed by inverted residual blocks that refine features progressively:32 → 16, 16 → 24, and 24 → 24 channels.

3.4. High-Level Features and ASPP Module. High-level semantic features are extracted and processed through the Atreus Spatial Pyramid Pooling (ASPP) module, which captures contextual information across multiple scales using:3×3 convolutions with dilation rates of 6, 12, and 18, and global image pooling for contextual understanding. These features are concatenated and projected through a 1×1 convolution to fuse the multi-scale context.

3.5. Decoder Module. The decoder refines segmentation predictions by merging high-level ASPP features with low level features. Key steps include: Concatenation of low- and high-level features to preserve both context and fine details. Further processing with 3×3 convolutions. Up sampling by a factor of 4 to match the input resolution.

3.6. Output Segmentation. The final output is a segmentation map, highlighting object boundaries and classes for each pixel.

4. Methodology

The methodology for building the semantic segmentation model follows a structured approach to ensure modularity, scalability, and efficiency. Each component of the pipeline is designed to handle specific tasks, from data preparation to model training and evaluation and the test unseen dataset. The process is divided into distinct phases, each building upon the previous ones to create a comprehensive system.

4.1 Data Processing and loading Methods. Data Loading: The next step was to load the dataset. I used the Pascal VOC dataset for segmentation. I created a custom dataset loader (VO Segmentation) to load both images and segmentation masks. This dataset class is based on PyTorch's Dataset class, allowing it to be easily used with Data Loader during training. Splitting the Dataset: The dataset was split into training, validation, and testing subsets. I used train_test_split: from scikit-learn to split the data.

4.2 The Principle of Model Design Architecture. Overview: The DeepLabV3+ architecture was used for the segmentation task. DeepLabV3+ is known for its effectiveness in semantic segmentation due to its Atreus Spatial Pyramid Pooling (ASPP) module, which helps in capturing multi-scale features. Backbone Selection: I used MobileNetV2 as the backbone because it's efficient and lightweight, making it suitable for edge devices and computationally constrained environments. MobileNetV2 uses depth wise separable convolutions, which reduces the computational load significantly. ASPP Module: The ASPP module employs dilated convolutions with different dilation rates to capture multi-scale context. This is crucial for segmentation tasks where context from various scales is necessary for accurate predictions.

Decoder and Output Layer: The decoder upscales the output of the ASPP module to the original resolution of the input image, providing a refined segmentation map. The output is then passed through a final convolutional layer to produce pixel-wise predictions.

4.3 Method of Adjusting Parameters.

Cross-Entropy Loss (nn.CrossEntropyLoss) was chosen as the loss function for this model due to its effectiveness and widespread use in semantic segmentation tasks. As a standard loss function for classification, it measures the difference between predicted probabilities and ground truth labels, making it well-suited for pixel-wise classification. The ignore index=255 parameter ensures that undefined or invalid labels (often marked as 255) are ignored during loss computation, while reduction='mean' averages the loss over all valid pixels, providing a stable and interpretable metric. Cross-Entropy Loss is computationally efficient, simple to implement, and inherently handles class imbalance, making it a reliable choice for this task.

The chosen learning rate of 0.01 is a standard starting point that balances the trade-off between convergence speed and stability. It allows the model to make steady progress in learning without overshooting the optimal point.

The polynomial decay schedule ensures a gradual reduction in learning rate, preventing sudden changes that could destabilize the training.

A batch size of 8 fits comfortably within GPU memory for high-resolution tasks like image segmentation and it ensures a good balance between computational efficiency and model accuracy.35,000 iterations, this value ensures the model has enough training time to converge effectively. Optimizer Momentum: 0.9, Weight Decay: 1e-4 SGD with momentum stabilizes the training process by dampening oscillations, ensuring steady convergence. • Weight decay reduces the risk of overfitting by adding regularization to the model parameters, leading to better generalization. Crop Size A crop size of 513 ensures that spatial details are preserved in the images, which is critical for segmentation tasks.

Validation Interval Every 100 epochs, Frequent validation provides timely feedback on model performance, allowing for better monitoring and early detection of potential overfitting. Checkpoints Checkpointing prevents loss of progress in case of interruptions, saving time and allows resumption of training.

```
# Train Options
parser.add_argument("--test_only", action='store_true', default=False)
parser.add_argument("--save_val_results", action='store_true', default=False,
                    help="save segmentation results to \"/content/drive/MyDrive/PROJECT/DeepLabV3Plus-Pytorch/r
parser.add_argument("--total_itrs", type=int, default=35000,
                    help="epoch number (default: 35000)")
parser.add_argument("--lr", type=float, default=0.01,
                    help="learning rate (default: 0.01)")
parser.add_argument("--lr_policy", type=str, default='poly', choices=['poly', 'step'],
                    help="learning rate scheduler policy")
parser.add_argument("--step_size", type=int, default=10000)
parser.add_argument("--crop_val", action='store_true', default=False,
                    help='crop validation (default: False)')
parser.add_argument("--batch_size", type=int, default=8,
                    help='batch size (default: 8)')
parser.add_argument("--val_batch_size", type=int, default=8,
                    help='batch size for validation (default: 8)')
parser.add_argument("--crop_size", type=int, default=513)

parser.add_argument("--ckpt", default=None, type=str,
                    help="restore from checkpoint")
parser.add_argument("--continue_training", action='store_true', default=False)

parser.add_argument("--loss_type", type=str, default='cross_entropy')
parser.add_argument("--gpu_id", type=str, default='0',
                    help="GPU ID")
parser.add_argument("--weight_decay", type=float, default=1e-4,
                    help='weight decay (default: 1e-4)')
parser.add_argument("--random_seed", type=int, default=1,
                    help="random seed (default: 1)")
parser.add_argument("--print_interval", type=int, default=10,
                    help="print interval of loss (default: 10)")
```

Figure 1. DeepLabV3+ training parameter configuration

5. Training and Evaluation the model

After creating all the required classes and parameters for training the model on the Pascal VOC 2012 dataset, it's now time to begin the training process. During training, we need to monitor the loss function after each iteration and the metrics after each epoch using TensorBoard for the entire training process.

```
#Train Loop
vis_sample_id = np.random.randint(0, len(val_loader), opts.vis_num_samples,
                                  np.int32) if opts.enable_vis else None  # sample idxs for visualization
denorm = utils.Denormalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  # denormalization for ori im

if opts.test_only:
    model.eval()
    val_score, ret_samples, val_loss = validate(
        opts=opts, model=model, loader=val_loader, device=device, metrics=metrics, criterion=criterion, ret_sam
    print(metrics.to_str(val_score))
    print(f"Validation Loss: {val_loss:.4f}")
    return

interval_loss = 0
best_score = 0.0
cur_itrs = 0
cur_epochs = 0
while True:  # cur_itrs < opts.total_itrs:
    # Train
    model.train()
    cur_epochs += 1

    # Reset metrics for each epoch
    metrics.reset()
    epoch_loss = 0.0
    epoch_samples = 0

    for (images, labels) in train_loader:
        cur_itrs += 1

        images = images.to(device, dtype=torch.float32)
        labels = labels.to(device, dtype=torch.long)
```

Figure 2. DeepLabV3+ training loop core code implementation

```
Training Metrics:

Overall Acc: 0.983950
Mean Acc: 0.973884
FreqW Acc: 0.969150
Mean IoU: 0.913910

Model saved as /content/drive/MyDrive/PROJECT/DeepLabV3Plus-Pytorch/checkpoints/latest_deeplabv3plus_mobilenet_voc_os16.pth
validation...
1449it [00:24, 58.23it/s]

Overall Acc: 0.914998
Mean Acc: 0.747995
FreqW Acc: 0.849659
Mean IoU: 0.646718

Validation Loss: 0.3222
Model saved as /content/drive/MyDrive/PROJECT/DeepLabV3Plus-Pytorch/checkpoints/best_deeplabv3plus_mobilenet_voc_os16.pth
Epoch 141, Itrs 25710/40000, Loss=0.049965
Epoch 141, Itrs 25720/40000, Loss=0.050647
```

Figure 3. Train/validate key metrics and model keeping records

As observed, after completing some epoch, the weights are saved in. path format for the last model, along with the best model saved separately as shown in the code below.

```
if (cur_itrs) % opts.val_interval == 0:
    save_ckpt('/content/drive/MyDrive/PROJECT/DeepLabV3Plus-Pytorch/checkpoints/latest_%s_%s_os%d.pth' %
              (opts.model, opts.dataset, opts.output_stride))
    metrics.reset()

    print("validation...")
    model.eval()
    val_score, ret_samples, val_loss = validate(
        opts=opts, model=model, loader=val_loader, device=device, metrics=metrics, criterion=criterion,
        ret_samples_ids=vis_sample_id)
    print(metrics.to_str(val_score))
    print(f"Validation Loss: {val_loss:.4f}")

    # Log validation metrics to TensorBoard
    writer.add_scalar('Loss/validation', val_loss, cur_itrs)  # Added validation loss logging
    writer.add_scalar('Metrics/Mean IoU', val_score['Mean IoU'], cur_itrs)
    writer.add_scalar('Metrics/Overall Acc', val_score['Overall Acc'], cur_itrs)

    if val_score['Mean IoU'] > best_score:  # save best model
        best_score = val_score['Mean IoU']
        save_ckpt('/content/drive/MyDrive/PROJECT/DeepLabV3Plus-Pytorch/checkpoints/best_%s_%s_os%d.pth' %
                  (opts.model, opts.dataset, opts.output_stride))
```

Figure 4. Verification phase process code implementation

This process continues iteratively to obtain the best possible result. The training is set to run for 35,000 iterations, equivalent to 192 epochs.

```
Epoch 191, Itrs 34910/40000, Loss=0.056950
Epoch 191, Itrs 34920/40000, Loss=0.041808
Epoch 191, Itrs 34930/40000, Loss=0.045262
Epoch 191, Itrs 34940/40000, Loss=0.040093
Epoch 191, Itrs 34950/40000, Loss=0.042566
Epoch 192, Itrs 34960/40000, Loss=0.049156
Epoch 192, Itrs 34970/40000, Loss=0.039534
Epoch 192, Itrs 34980/40000, Loss=0.045228
Epoch 192, Itrs 34990/40000, Loss=0.045057
Epoch 192, Itrs 35000/40000, Loss=0.043004
183it [00:18,  9.96it/s]
Training Metrics:

Overall Acc: 0.986168
Mean Acc: 0.976339
FreqW Acc: 0.973285
Mean IoU: 0.925057

Model saved as /content/drive/MyDrive/PROJECT/DeepLabV3Plus-Pytorch/checkpoints/latest_deeplabv3plus_mobilenet_voc_os16.pth
validation...
1449it [00:25, 56.91it/s]

Overall Acc: 0.915719
Mean Acc: 0.738104
FreqW Acc: 0.850121
Mean IoU: 0.645294

Validation Loss: 0.3211
```

Figure 5. Real-time log of training progress

After finishing all epochs, TensorBoard is used to review all the metrics recorded throughout the entire training period such as loss function curves and accuracy curves and Mean IoU for training and validation as shown below.

6.Result

In this section, present a detailed analysis of the results obtained after running the semantic segmentation model. The output from the model is stored in the form of single-channel grayscale masks, which are typically used for binary classification tasks, where each pixel is assigned, a value representing its classification. These grayscale masks allow us to visualize the model's segmentation output effectively, as each pixel intensity corresponds to a specific class or category predicted by the model. The following random images, including both the single- channel grayscale masks and their corresponding original images.

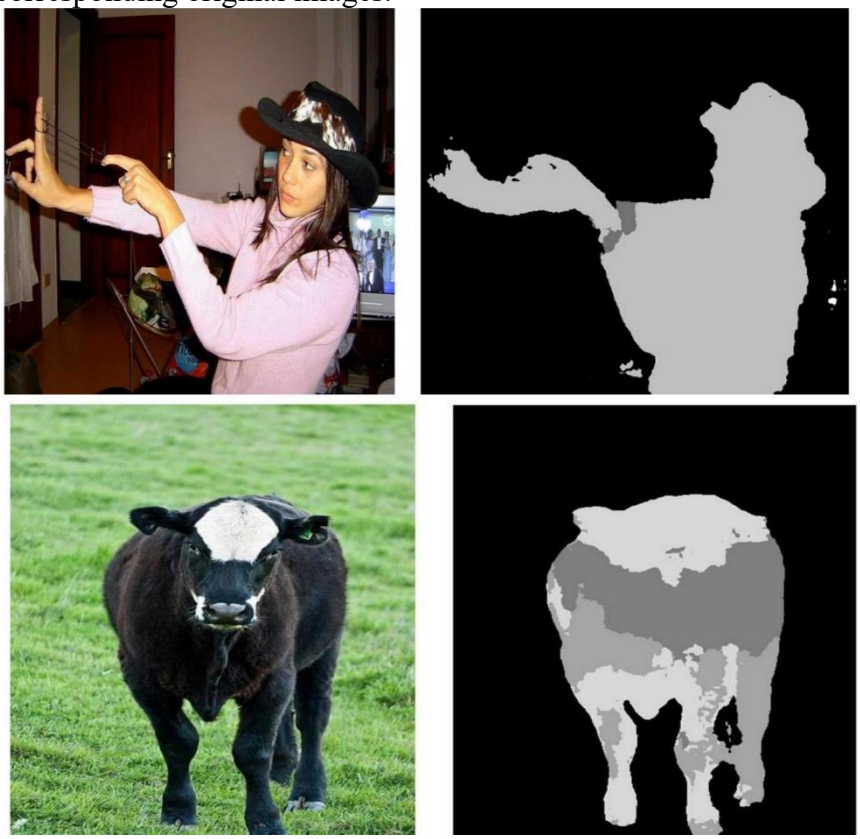

Figure 6. Qualitative comparison of semantic segmentation results

7. Conclusion

This project successfully implemented a semantic segmentation model using the DeepLabV3+ architecture with a MobileNetV2 backbone, applied to the widely used Pascal VOC2012 dataset. The goal was to develop a lightweight, efficient, and accurate model capable of assigning semantic class labels to each pixel in an image—a task that is both fundamental and challenging in computer vision.

Throughout the project, a well-structured methodology was followed, starting from dataset preparation and environment setup, to model design, training, and final evaluation. A custom data loading pipeline was implemented to manage image and mask pairing, along with appropriate transformations and augmentations. The DeepLabV3+ architecture, known for its powerful segmentation capabilities, was enhanced with the MobileNetV2 backbone to ensure efficiency in terms of speed and resource usage—an important consideration for real-time and edge computing applications.

The training process, conducted in Google Colab with TensorBoard integration, showed steady improvement across metrics such as loss and accuracy. The training loss decreased significantly from 1.89 to below 0.08, while the validation loss also showed consistent improvement. The final model achieved a high training accuracy of 98.5% and validation accuracy of approximately 91.5%, indicating strong generalization to unseen data. Furthermore, the model effectively generated grayscale mask predictions on the test dataset, visually confirming the accuracy of object segmentation across diverse classes.

The model's use of the Atreus Spatial Pyramid Pooling (ASPP) module enabled it to capture multi-scale context, which is crucial for segmenting objects of varying sizes and complexities. The decoder module further refined the segmentation output by integrating low-level and high- level features, improving boundary precision.

In conclusion, this project not only met its technical objectives but also provided a deep understanding of the entire semantic segmentation pipeline. It highlighted the importance of model architecture selection, training optimization, and evaluation techniques. The resulting system is robust, scalable, and suitable for real-world applications, including autonomous vehicles, medical imaging, and smart city systems. Future improvements could involve experimenting with different backbones, applying advanced data augmentation techniques, or exploring post-processing methods like Conditional Random Fields (CRFs) to further enhance segmentation accuracy.

References

[1] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2018).DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(4), 834–848.

[2] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4510–4520.

[3] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, 88(2), 303–338.

[4] PyTorch Documentation.https://pytorch.org/docs/stable/index.html