Real Time Semantic Segmentation

Kashif Ansari
University of Maryland
kansari@umd.edu

Sachin Jadhav
University of Maryland
sjd3333@umd.edu

Kautilya Reddy
University of Maryland
nsingh19@umd.edu

Navdeep Singh
University of Maryland
kautilya@umd.edu

Abstract

This paper presents an in-depth study on implementing real-time semantic segmentation in autonomous robotics, focusing on enhancing perception for tasks such as object detection, navigation, and human-robot interaction. We investigate the performance of two prominent deep learning architectures, U-Net and DeepLabV3+, across various datasets, including CamVid and Human Parsing, with the goal of achieving high accuracy and efficient processing. Our methodology involves comprehensive steps from data collection and preprocessing to model selection, training, optimization, and integration with the Luxonis OAK-D camera for real-time applications. The DeepLabV3+ model emerged as the superior choice, demonstrating higher validation accuracy and Intersection over Union (IoU) scores compared to U-Net. This paper also highlights the challenges of handling noise in image frames and optimizing model performance using TensorFlow and OpenVINO tools. Our results underscore the importance of robust real-time semantic segmentation in advancing autonomous systems, providing a significant contribution to the field of machine perception.

1. Introduction

Please follow the steps outlined below when submitting your manuscript to the IEEE Computer Society Press. This style guide now has several important modifications (for example, you are no longer warned against the use of sticky tape to attach your artwork to the paper), so all authors should read this new version.

1.1. Background

In the realm of autonomous robotics, perception is pivotal for machines to effectively understand and interact with their environment. Semantic segmentation, a fundamental task in perception, assigns semantic labels to image pixels, segmenting them into meaningful regions based on objects or classes. Real-time semantic segmentation is a critical advancement, providing timely and accurate segmentation crucial for decision-making in autonomous systems. These algorithms operate swiftly and efficiently, addressing the challenge of balancing accuracy and speed using deep learning architectures like CNNs optimized for rapid processing. Integrating real-time semantic segmentation into autonomous robots enables real-time perception, enhancing tasks such as object detection, navigation, and human-robot interaction, especially in scenarios like autonomous driving where it identifies road elements for safer navigation. Advancements in sensor technologies further complement real-time semantic segmentation by providing rich data for accurate segmentation and scene analysis, enhancing algorithm reliability in challenging environmental conditions.

1.2. Robotics implementations

Autonomous robots drive technological innovation by autonomously perceiving, understanding, and navigating complex environments, relying on advanced techniques like real-time semantic segmentation for precise and swift environmental perception. Semantic segmentation categorizes each pixel in an image into meaningful classes, enabling detailed scene understanding and informed decision-making for robots, crucial in applications requiring rapid decision-making like autonomous driving, surveillance, navigation, and human-robot collaboration, ensuring safety, efficiency, and effective interaction. Challenges in real-time semantic segmentation involve achieving high accuracy while processing large visual data volumes efficiently, balancing precision with computational speed. Modern real-time semantic segmentation leverages deep learning, particularly CNNs, optimized for speed through model compression, lightweight designs, and hardware acceleration on resource-limited platforms. Integrating real-time semantic segmentation enhances robots' perceptual abilities, aiding navigation, obstacle avoidance, task execution, and adaptive behavior based on environmental context.

1.3. Steps we followed for Building this project

Steps in Building Real-time Semantic Segmentation from Scratch

1. **Data Collection and Annotation:**
   - Collect a diverse dataset of images relevant to the target application.
   - Annotate the dataset with semantic labels for each pixel.
2. **Data Preprocessing:**
   - Resize and normalize images to a standardized format.
   - Augment the dataset with techniques like rotation and flipping.
3. **Model Selection:**
   - Choose a deep learning architecture suitable for semantic segmentation.
   - Customize the architecture based on computational resources.
4. **Model Training:**
   - Split the dataset into training, validation, and test sets.
   - Train the model using appropriate loss functions.
5. **Optimization Techniques:**
   - Implement optimization techniques like gradient clipping and regularization.
   - Explore model compression techniques to reduce size and improve speed.
6. **Inference Pipeline Development:**
   - Develop an inference pipeline for real-time segmentation.
   - Optimize the pipeline for efficiency and GPU acceleration.
7. **Performance Evaluation:**
   - Evaluate the model using metrics like Intersection over Union (IoU).
   - Analyze performance in terms of accuracy and speed.
8. **Integration with Robotics Platform:**
   - Integrate the model with the robotics platform.
   - Validate performance in real-world scenarios.
9. **Continuous Improvement:**
   - Monitor and update the model based on feedback.
   - Incorporate advancements in deep learning research.

2. Methodology

We used two datasets for training on two different back bones. One dataset was the car segmentation dataset and the CamVid dataset. These datasets consisted of 5 classes and 12 classes respectively for segmentation task. Upon performance evaluation we finally narrowed down to the DeepLabV3Plus model for training on the Human Parsing dataset. Another reason for selecting this model was that the results produced were better reproducible with Luxinious Camera.

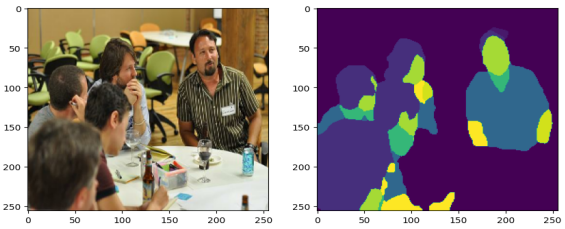

Figure 1. Human Parsing Datasets

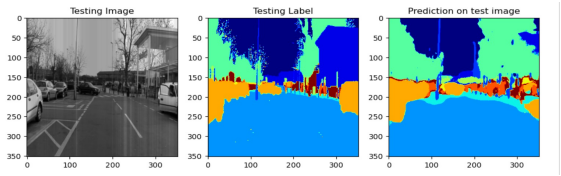

Figure 2. CamVid Datset

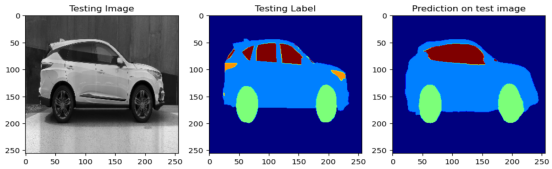

Figure 3. Vehicle Dataset

2.1. Backbone for Image Segmentation

We tested with two Real Time Segmentation Datasets using backbones ”UNet” and ”DeepLab V3+”.

Tensorflow was chosen as the implementation architecture due to the ease of configuring the weights. We downscaled the images to a size of (256x256) and used a batch-size of 16 to ensure easy processing based on the system capabilities.

The Datset was split into Training and Validation with a 0.1 split. We implemented Label Encoding with class weights to hasten the process of training as it encodes the identified segmentation with the relevant classes in a binary matrix format as tensors. Later, the results were further tested on a separate test

dataset.

This was followed After implementing the Convolution, Thresholding and Maxpooling Layers for the backbone Structure. Training with the respective architectures then provided us with model weights in ONNX (Open Neural Network Exchange) format which would then be converted to the Blob format to make it recognisible by the Luxinous.

We constructed a script that implemented the model results on each frame and relayed the model result to produce Real Time Semantic Segmnetation. This was implemented on ou

2.2. Comparision between UNet and Deep Lab V3+ Architectures

Both U-Net and DeepLabV3+ are powerful architectures for image segmentation, each with its unique strengths. U-Net's simplicity and effectiveness in retaining spatial details make it ideal for medical imaging, while DeepLabV3+'s ability to capture multi-scale context and handle complex segmentation tasks makes it a superior choice for more general-purpose applications.

As a result, DeepLabV3+ should be a more appropriate choice for training on Human parsing dataset as it consists of multiple classes. This is also evident from the numerical results , that is a Validation Accuracy of 0.95 and an IOU Score of 0.7 which represents a superior performance of the model.

DeepLab V3+ Architecture

1. Encoder:
   - **Convolutional layers:**Uses a pre-trained backbone like ResNet or Xception, which includes multiple convolutional layers with ReLU activations and batch normalization.
   - **Atrous Convolutions:**In the later stages, standard convolutions are replaced with atrous (dilated) convolutions to maintain spatial resolution and capture multi-scale context without downsampling.
   - **Feature Extraction:** Provides high-level features with reduced spatial dimensions but retains more context information compared to traditional pooling operations.
2. Atrous Spatial Pyramid Pooling (ASPP):
   - **Parallel Atrous Convolutions:** Consists of multiple parallel atrous convolutions with different dilation rates (e.g., 1, 6, 12, 18) to capture multi-scale information.
   - **Global Average Pooling:** Includes a global average pooling layer that captures global context information.
   - **1x1 Convolutional** Applies a 1x1 convolution to reduce the number of feature channels after concatenation of the atrous convolutions.
   - **Batch Normalization and ReLU:** Each convolution in the ASPP module is followed by batch normalization and ReLU activation.
3. Decoder Path:
   - **Low-Level Feature Concatenation:**Low-level features from earlier stages of the backbone are concatenated with the upsampled ASPP output.
   - **1x1 Convolution:** Reduces the number of channels in the low-level features before concatenation.
   - **Upsampling Layers:** Several 3x3 convolutions followed by bilinear upsampling to refine the segmentation map.
   - **Final Unsampling:** The final output is upsampled to the original image resolution.
4. Final Layers:
   - **1x1 Convolution:** Similar to U-Net, a 1x1 convolution maps the output to the number of classes for segmentation.

Layer Details:

- **Activation:** ReLU activations are used throughout the network.
- **Normalization:** Batch normalization is consistently applied after convolutions to ensure stable training and better performance.

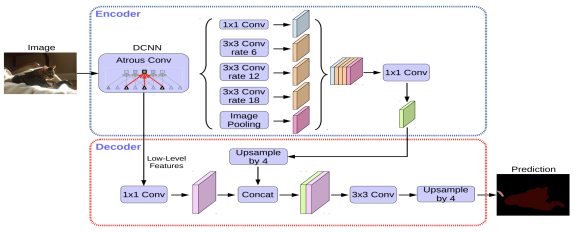

Figure 4. Deep LabV3+ Architecture

As a result, the Human Parsing Dataset was well trained using DeepLabV3+ and provided an outstanding Validation Accuracy of 0.9 and an IOU Score of 0.7 which respresents a superior performance of the model.

Unet Architecture

1. Encoder Path:

- **Convolutional layers:**Each level in the encoder consists of two 3x3 convolutional layers followed by a ReLU activation function. This doubles the number of feature channels.
- **Pooling Layers:**After the convolutions, a 2x2 max-pooling operation with a stride of 2 is applied, reducing the spatial dimensions by half.
- **Feature Channels:**Typically, the number of channels starts at a base value (e.g., 64) and doubles after each downsampling step.

2. Bottleneck:

   - **Convolutional Layers:** At the bottleneck, two 3x3 convolutional layers with ReLU activations are used, similar to other encoder levels, without downsampling.
   - **Feature Channels:** This layer has the highest number of feature channels (e.g., 1024).

3. Decoder Path:

   - **Upsampling Layers:**Each level in the decoder begins with a 2x2 up-convolution (transposed convolution) that halves the number of feature channels and doubles the spatial dimensions.
   - **Skip Connections:** The output of the up-convolution is concatenated with the corresponding feature map from the encoder path.
   - **Convolutional Layers:** Following the concatenation, two 3x3 convolutional layers with ReLU activations are applied, reducing the number of feature channels.

4. Final Layers:

   - **1x1 Convolution:** A final 1x1 convolution is applied to map the output to the desired number of classes for segmentation.

Layer Details:

- **Activation:** ReLU is used after each convolutional layer.
- **Normalization:** Batch normalization is often added in modified versions of U-Net for better performance.

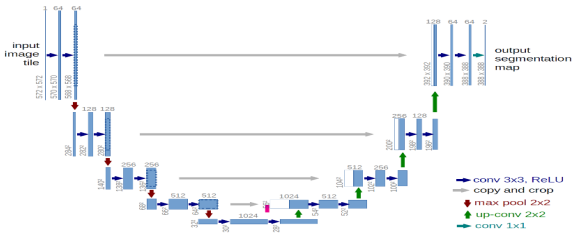

Figure 5. U-Net Architecture

2.3. Hardware Implementation

The used Hardware for this project is Luxonis OAK-D camera which is an advanced depth-sensing camera designed for a variety of applications, including robotics, computer vision, and AI-driven projects. Featuring a triple-camera setup, the OAK-D combines a 12.3 MP RGB camera with dual monochrome global shutter cameras to deliver accurate depth perception and high-quality imaging. This powerful device leverages Intel's Movidius Myriad X VPU, enabling on-device neural network processing for real-time object detection, tracking, and depth estimation. With robust performance and versatility, the OAK-D is suitable for developers and researchers looking to integrate sophisticated visual AI capabilities into their systems, facilitating innovative solutions in autonomous vehicles, industrial automation, and more.

Camera Specs	Color camera	Stereo pair
Sensor	IMX378 (PY011 AF)	OV9282 (PY010 FF)
DFOV / HFOV / VFOV	81° / 69° / 55°	81° / 72° / 49°
Resolution	12MP (4056x3040)	1MP (1280x800)
Focus	AF: 8cm - or FF: 50cm - inf	FF: 19.6cm - inf
Max Framerate	60 FPS	120 FPS
F-number	1.8±5%	2.0±5%
Lens size	1/2.3 inch	1/4 inch
Effective Focal Length	4.81mm	2.35mm
Pixel size	1.55µm x 1.55µm	3µm x 3µm

Table 1. Camera Specifications

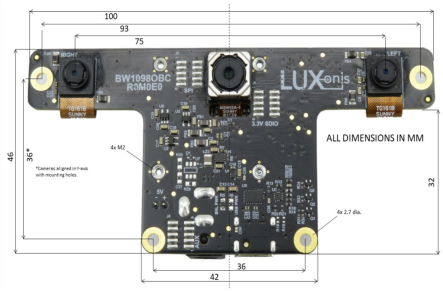

Figure 6. Luxinous Camera

Fig. 5. Illustrates the process of transferring a deep learning model for semantic segmentation to a camera with onboard computation capabilities.

1. **Model**: The process begins with a pre-trained deep learning model. This model could be developed using various frameworks such as TensorFlow or PyTorch.

2. **Onnx (Open Neural Network Exchange)**: The model is then converted to the ONNX format, which provides interoperability between different deep learning frameworks. This step ensures that the model can be easily integrated into the next stages of the pipeline.

3. **OpenVINO**: Once in the ONNX format, the model is processed using Intel's OpenVINO toolkit. OpenVINO optimizes the model for inference, taking advantage of Intel hardware capabilities to improve performance and efficiency.

4. **OpenVINO IR (Intermediate Representation)**: The OpenVINO toolkit converts the optimized model into an Intermediate Representation (IR). The IR is a pair of files (.xml and .bin) that contains the model structure and the associated weights, optimized for inference on various Intel hardware.

5. **Blob Converter**: The IR files are then passed through a Blob Converter. This tool converts the IR format into a blob, which is a binary file format specifically designed for execution on the camera's hardware.

6. **Blob**: The resulting blob file contains the optimized model in a format that the camera's onboard computational unit can understand and execute.

7. **Camera**: Finally, the blob is loaded onto the camera. The camera, equipped with onboard computation capabilities, can now perform semantic segmentation in real-time, leveraging the optimized deep learning model for efficient and accurate inference.

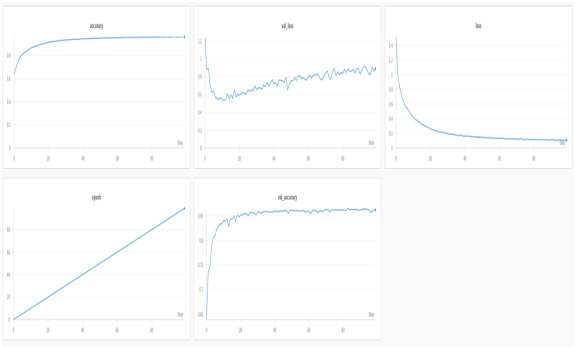

Figure 7. DeepLab V3+ Results

Figure 8. Real Time Segmentation results

3. Results

This pipeline ensures that the deep learning model is efficiently transferred and optimized for the camera, enabling high-performance semantic segmentation directly on the device.

Several experiments were carried out for testing the two models namely DeepLabV3Plus and the U-Net model and the results were evaluated on several datasets. CamVid was a challenging dataset that was utilized for testing. This dataset consists of street views taken from self driving cars and provides ground truth masks which consist of segmentation for various objects in the street view. The idea behind the development of this dataset was to encourage the development of multi-class semantic segmentation algorithms for self driving cars and through these algorithms various elements in the street view such as the other cars, pedestrians, etc. could be recognized and their precise locations determined as this would further aid in the development of various machine learning based or algorithmic based decisions making programs for self driving cars.

This challenging dataset consists of 12 different object classes for segmentation. The results achieved for the two models when trained on this dataset were a final validation accuracy of 0.80 for the UNet model and a final validation accuracy of 0.85 for the DeepLabV3Plus model.

Additionally, because we are representing all the classes for human parsing by disabling the color matrix, the real-time segmentation displays all classes in the same color. This limitation can be addressed by modifying the real-time segmentation code to enable distinct color representation for each class. Importantly, the uniform color display does not indicate a flaw in the model itself, as the accurate segmentation results from the test images demonstrate the model's reliability and effectiveness.

Also the average IoU value achieved for the UNet model is 0.35 and that for DeepLabV3Plus was 0.40. Therefore it was clear through these metrics that the DeepLabV3Plus model was a better choice out of the two models for training on the Human Parsing dataset which consisted of 20 different object classes for segmentation. The validation accuracy achieved for DeepLabV3 model for the Human Parsing dataset was 0.95 which implies a superior performance by the model.

4. References

- U-Net: Research Paper
- Hardware Used: OAK-D Camera
- Data Set Used: Car Segmentation Dataset
- Our complete Repository: Realtime Segmentation Repository