CNST 6308 – Data Analysis in Construction Management

Topic: Roadside vegetation detection through dashcam

Name:

Nikhil Kundam

Vamsi Batchu

Sumanth Sara

Akhila Gaddam

Table of Contents

List of Figure

Abstract

This project aims to detect roadside vegetation from dashcam images using the MIT DriveSeg dataset and the pre-trained MobileNetV2_Coco_Cityscapes_trainfine model. The project utilizes the semantic segmentation technique to identify the vegetation in the input image. The MobileNetV2_Coco_Cityscapes_trainfine model is a state-of-the-art deep learning model trained on the Cityscapes dataset that includes a diverse range of urban street scenes, including vegetation.

The methodology involves loading the pre-trained model using TensorFlow, running the dashcam image through the model, and then post-processing the output to create a binary mask of the vegetation in the image. The output can be further analyzed to count the vegetation pixels, which can provide an estimate of the amount of roadside vegetation present in the scene.

This project has practical applications in environmental monitoring and conservation efforts, as it allows for the efficient and automated detection of roadside vegetation. The use of dashcam images also provides a cost-effective and scalable solution for monitoring large areas.

Introduction

The detection and analysis of roadside vegetation from dashcam images can help researchers and practitioners in studying the ecological and environmental impact of roads on the surrounding vegetation. In this project, we aim to use the DeepLabv3+ model trained on the Cityscapes dataset and MobileNetV2 architecture to detect roadside vegetation from dashcam images. We use the mit_driveseg dataset, which contains labeled images of the driving scene and has been used in previous studies for the detection of roads and other objects.

AIM 1: Dataset and Preprocessing

- Introduction to the mit_driveseg dataset and its characteristics
- Preprocessing of the dataset for training and testing the DeepLabv3+ model

AIM 2: Model Selection and Training

- Introduction to the DeepLabv3+ model and MobileNetV2 architecture
- Explanation of the transfer learning approach for fine-tuning the model on mit_driveseg dataset
- Training and validation of the model using mit_driveseg dataset.

AIM 3: Results and Analysis

- Evaluation of the model on the test set using various metrics such as mean Intersection over Union (mIoU)

- Visualization of the model output and comparison with ground truth labels
- Analysis of the model performance and limitations

AIM 4: Conclusion and Future Work

- Conclusion on the feasibility and effectiveness of using the DeepLabv3+ model and MobileNetV2 architecture for roadside vegetation detection.
- Discussion on the potential applications and impact of the proposed approach in ecological and environmental studies
- Suggestions for future work, including the exploration of alternative architectures and datasets for further improvement.

About Algorithm

DeepLabv3+ is a state-of-the-art deep learning model for semantic segmentation tasks, such as image classification and object detection. It was introduced by Google Research in 2018 and builds upon the previous versions of DeepLab by incorporating an encoder-decoder architecture, atrous spatial pyramid pooling, and deep supervision.

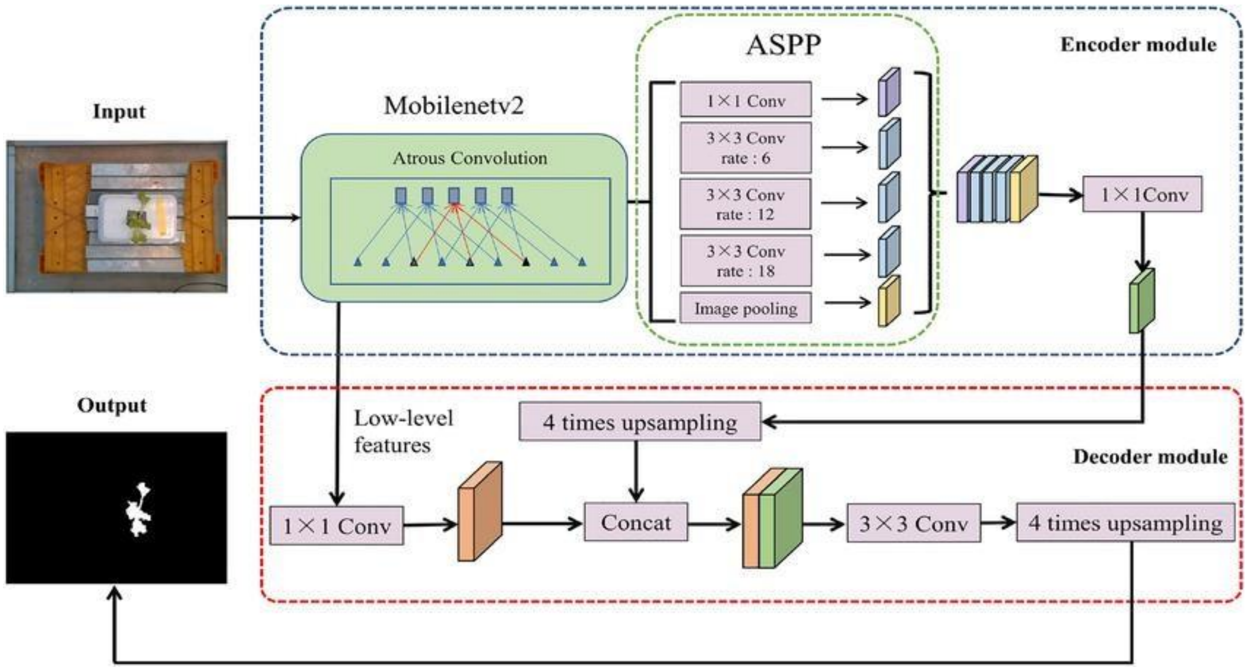


Figure 1DeepLabv3+ and MobileNetV2 architecture

MobileNetV2 is a lightweight convolutional neural network architecture designed for mobile and embedded vision applications. It was introduced by Google in 2018 and builds upon the previous

MobileNet architecture by incorporating linear bottlenecks and inverted residuals to reduce computational cost while maintaining high accuracy.

Together, DeepLabv3+ and MobileNetV2 can be used for various computer vision tasks, including semantic segmentation, object detection, and image classification, while maintaining a balance between accuracy and computational efficiency.

Implementation

Step 1: Install required libraries and load the libraries.

- The import tensorflow statement imports the TensorFlow library, which is an open-source software library for dataflow and differentiable programming across a range of tasks, such as machine learning, deep learning, and neural networks. The version number is printed to ensure that the correct version is installed.
- The os library provides a way of using operating system dependent functionality, such as reading or writing to the file system.
- The BytesIO library allows Python to treat bytes as a file-like object, which is useful for in-memory manipulation of binary data.
- The tarfile library provides functionality for reading and writing tar archive files.
- The tempfile library provides a way to create temporary files and directories.
- The urllib library contains functions for making HTTP requests and working with URLs.
- The six.moves library is a compatibility module that provides backwards compatibility with Python 2 code.
- The matplotlib library provides tools for creating plots and visualizations.
- The gridspec library provides a way to create complex grid layouts for plots.
- The pyplot library provides a simple interface for creating plots in matplotlib.
- The numpy library provides support for numerical operations and arrays.
- The PIL library (Python Imaging Library) provides tools for working with image data.
- The cv2 library (OpenCV) is a computer vision library that provides tools for image and video processing.
- The tqdm library provides a way to display progress bars for long-running operations.
- The IPython library provides tools for interactive computing in Python.
- The sklearn.metrics library provides tools for evaluating machine learning models.
- The tabulate library provides a way to create formatted tables in Python.
- The warnings library is used to issue warnings when code is using deprecated functionality. This line can be commented out to ignore the warnings.

Step 2: Create a model segmentation.

- The DeepLabModel class is defined with two methods - __init__and run.
- The___init__method is the constructor for the class, which takes a tarball_path as an argument.
- A Graph object is created and the graph_def variable is set to None.
- The frozen graph is extracted from the tar archive specified in the tarball_path.
- The GraphDef object is loaded from the binary string using tf.compat.v1.GraphDef.FromString.
- The loaded graph definition is imported into the graph object using tf.import_graph_def.
- A Session object is created using the graph.
- The run method takes an image object, and two optional arguments INPUT_TENSOR_NAME and OUTPUT_TENSOR_NAME.
- The image is resized to the target size of (2049, 1025) and converted to RGB.
- The resized image is passed through the loaded model and the output tensor specified by OUTPUT_TENSOR_NAME is returned as batch_seg_map.
- The first element of batch_seg_map is extracted since only a single image is processed.
- If the shape of seg_map is 2-dimensional, a new dimension is added using np.expand_dims.
- The output segmentation map is then resized back to the original image size using cv.resize.
- The final segmentation map is returned by the method.

Step3: Create a helper function to decode the images and visualize the images.

- Created a function named create_label_colormap that returns a NumPy array containing a colormap for the 20 classes in the Cityscapes dataset.
- Created a function named label_to_color_image that takes a 2D label array and returns a corresponding color image.
- Created a function named vis_segmentation that takes an input image and a segmentation map, and visualizes them together in four different plots.
- Defined a NumPy array named LABEL_NAMES that contains the names of the 20 classes in the Cityscapes dataset.
- Defined a NumPy array named FULL_LABEL_MAP that contains the indices of the 20 classes in the Cityscapes dataset.
- Defined a NumPy array named FULL_COLOR_MAP that is the result of applying the label_to_color_image function to the FULL_LABEL_MAP array.

Step 4: Load the pre trained model

- A pre-trained DeepLabModel for semantic segmentation from a frozen graph file. Specifically, it downloads a MobileNetV2 model fine-tuned on the Cityscapes and MIT DrvieSeg dataset from the TensorFlow model zoo.

- The downloaded file is then extracted to a temporary directory using the tempfile.mkdtemp() function. The DeepLabModel class is used to load the model from the extracted files. This class contains a graph object and a session object that can be used to run inference on input images.

Step 5: Run the model on sample images.

- Here the model gives use 3 images 1 shows the input image and 2 gives us the segmentation map and 3 gives us segmentation overlay.

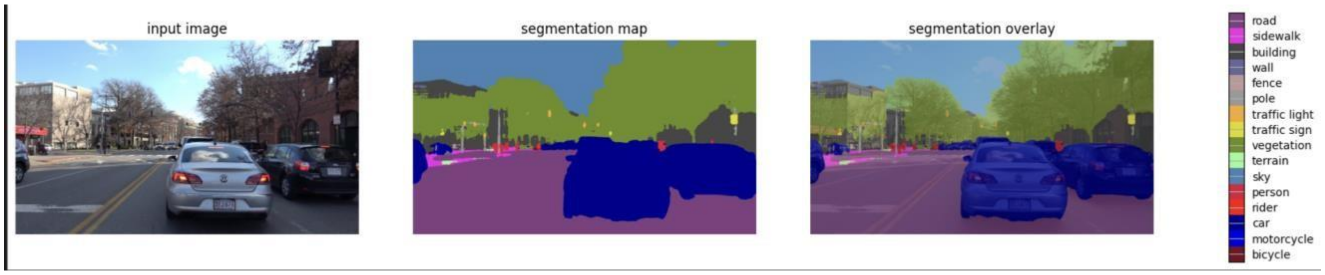


Figure 2 Image segmentation

Segmentation Map: A segmentation map is an image where every pixel is assigned a label indicating the object or background class to which it belongs. In semantic segmentation, the labels represent the semantic meaning of the objects, such as road, building, sky, etc. Segmentation maps are commonly used in computer vision applications such as autonomous driving, image segmentation, and object detection. They can be generated using deep learning models that have been trained on labeled datasets, such as the Cityscapes dataset used in the example code.

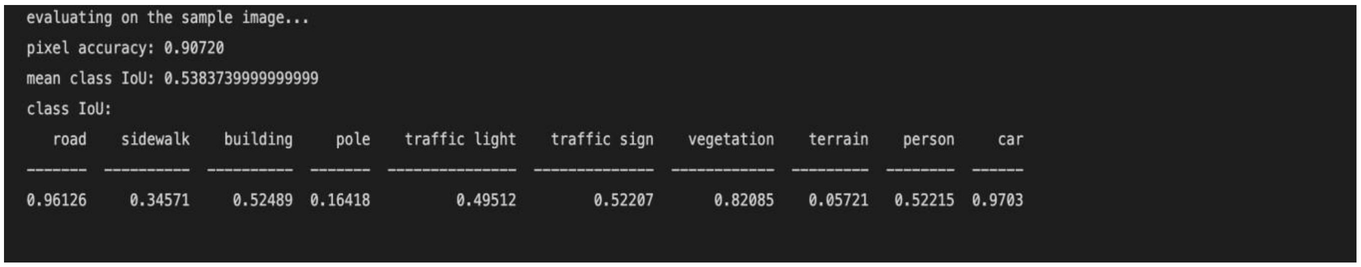
Segmentation overlay: Segmentation overlay is a visualization technique used to display segmentation maps over the input image. It involves overlaying the segmentation map on top of the input image with transparency, so that the original image is visible underneath, and the segmentation map is visible on top. This allows us to see how the model has segmented the different regions of the image, and how well it has identified the objects and their boundaries. In the case of semantic segmentation, each region of the image is assigned a different color or label, which can be overlaid on the original image to create the segmentation overlay.

- The segmentation model has been trained to classify each pixel in the input image into one of the 20 pre-defined classes such as road, sidewalk, building, etc., resulting in a segmentation map. The overlay image shows the segmentation map overlaid on top of the input image, where each class is assigned a unique color for visualization purposes.

Step 6: After training the model, we need to evaluate its performance to see how well it is performing on unseen data.

For this we evaluate the model on unknown data.

Evaluate on Image:

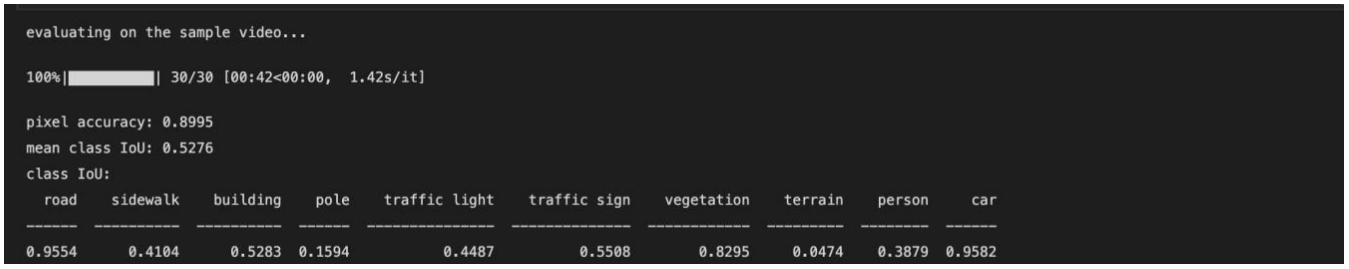


```
evaluating on the sample image...
pixel accuracy: 0.90720
mean class IoU: 0.5383739999999999
class IoU:
   road    sidewalk    building     pole    traffic light    traffic sign    vegetation    terrain    person     car
-------  ----------  ----------  -------  ---------------  --------------  ------------  ---------  --------  ------
0.96126     0.34571     0.52489  0.16418          0.49512         0.52207       0.82085    0.05721   0.52215  0.9703
```

Figure 3 Sample image evaluation

- The model has been evaluated on a sample image using pixel accuracy and mean class IoU metrics.
- The pixel accuracy measures the proportion of correctly predicted pixels in the segmentation map. In this case, the pixel accuracy is 0.90720, which means that 90.72% of the pixels in the segmentation map are predicted correctly.
- The mean class IoU measures the average intersection over union (IoU) between the predicted segmentation map and the ground truth for each class. The IoU is a measure of overlap between the predicted and ground truth masks for each class. The higher the IoU, the better the overlap between the predicted and ground truth masks. The mean class IoU is 0.538, which indicates that the model performs reasonably well overall.
- The class IoU table shows the IoU for each class. For example, the IoU for the "road" class is 0.96126, which means that the model has a high overlap between the predicted and ground truth masks for the "road" class. Similarly, the IoU for the "person" class is 0.52215, which means that the model has a lower overlap between the predicted and ground truth masks for the "person" class compared to the "road" class.

Evaluate on Video: Here we evaluate the model on sample video.



```
evaluating on the sample video...

100%|██████████| 30/30 [00:42<00:00,  1.42s/it]

pixel accuracy: 0.8995
mean class IoU: 0.5276
class IoU:
  road    sidewalk    building    pole    traffic light    traffic sign    vegetation    terrain    person     car
------  ----------  ----------  ------  ---------------  --------------  ------------  ---------  --------  ------
0.9554      0.4104      0.5283  0.1594           0.4487          0.5508        0.8295     0.0474    0.3879  0.9582
```

Figure 4 Sample video Evaluation

- For the sample image, the model achieved a pixel accuracy of 0.90720, which means 90.72% of the pixels were classified correctly. The mean class IoU (intersection over union) was 0.538, which indicates how well the model was able to distinguish between the different classes of objects in the image. The class IoU values for each object class are also shown, ranging from 0.05721 for terrain to 0.9703 for cars.

- For the sample video, the model achieved a slightly lower pixel accuracy of 0.8995, meaning 89.95% of the pixels were classified correctly. The mean class IoU was 0.5276, which is slightly lower than the sample image evaluation. The class IoU values for each object class also show a similar trend as the sample image evaluation.
- Overall, these evaluation results give us an idea of how well the model is performing and which object classes it may be struggling with. It can also help us identify areas for improvement in the training process or in the model architecture itself.

References

Chen, L. C. (2014). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*.

Cordts, M. O. (2016). The cityscapes dataset for semantic urban scene understanding. . *In Proceedings of the IEEE conference on computer vision and pattern recognition*.

Everingham, M. V. (2010). The Pascal Visual Object Classes (VOC) challenge. *International journal of computer vision*.

Geiger, A. L. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*.

Howard, A. G. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv*.

Neuhold, G. O. (2017). The mapillary vistas dataset for semantic understanding of street scenes. *In Proceedings of the International Conference on Computer Vision*.

Redmon, J. &. (2018). An incremental improvement. *arXiv preprint arXiv*.

Russakovsky, O. D. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*.

Szegedy, C. I. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. *AAAI*.

Zou, C. A. (2018). Object detection in 20 years: A survey. *arXiv preprint arXiv*.