Fine-tuning Cityscape on DeepLabv3 Semantic Segmentation model

Loïc Duchesne
McGill University
Faculty of Engineering
Montreal, Canada
loic.duchesne@mail.mcgill.ca

***Abstract*—With the continuous progression in computer vision, semantic segmentation has become an essential technique in understanding and interpreting visual data. However, achieving high accuracy in diverse and complex environments remains difficult. This project explores the fine-tuning of semantic segmentation models using the DeepLabv3 architecture with a ResNet-50 backbone. By leveraging this advanced model, we demonstrate its ability to effectively segment and classify objects within images by fine-tuning the current weights, showcasing significant improvements compared to its original model. This study aims to highlight the potential of fine-tuning DeepLabv3 ResNet-50 in improving domain-specific image analysis tasks.**

***Index Terms*—Semantic Segmentation, Deep Learning, Convolutional Network, Computer Vision**

I. Introduction

With significant advancement in the artificial intelligence field, the AI models are becoming increasingly bigger in sizes and parameters. These models hold significant complexity and machine intelligence, but may sometimes lack in certain domains. Despite that, their advanced complexity allows them to be trained on new information, and drastically outperform models trained from scratch while saving significant computing cost. In this paper, I will demonstrate the ability to improve such models on new data and allow it to reach new peaks in domain specific intelligence.

II. Architecture/Algorithm

A. Dataset

1) Dataset generation: For this experiment, we are using the Cityscapes dataset. We are using leftImg8bit and gtFine from that dataset which provides us with the original images and the labels. We did have to generate the training labels as they were not provided, but the generatetrainIDs from cityscapesscripts.py generated them afterwards.

Fig. 1. One of the original images from Aachen_19_leftImg8bit

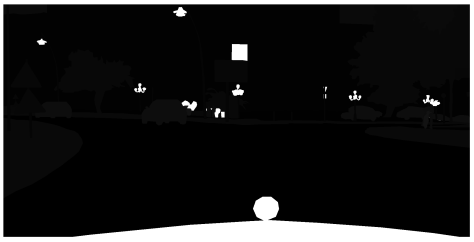

Fig. 2. One of the labelled training IDs images from Aachen_19_gtFine

As we can see in the image below, it is extremely difficult to distinguish the different objects. This is due to all the relevant objects being adjusted to their respective classes, while displaying an image expecting 255. Note the bright white objects, which are objects that are excluded from the training process. Therefore, their pixel have been assigned a value of 255.

2) Pre-processing: To feed the images to the model we wish to train, we must process them using either PyTorch's pre-processing framework or in our case, used the Albumentations framework which provides more rigorous methods for semantic segmentation purposes. The images were processed with the following transforms:

```
A.Resize(HEIGHT, WIDTH),
A.HorizontalFlip(),
A.Normalize(mean=(0.485, 0.456, 0.406),
std=(0.229, 0.224, 0.225),
max_pixel_value=255.0),
ToTensorV2()
```

The image was firstly resized to a certain height a width to ensure all images are the same size. The specific height and width is not specified as it is mostly dependent on the hardware available, and will be investigated in a further section. Afterwards, depending if it is a training or validation transform, the image is flipped to improve training performance. It is then normalized between 0 and 1 using ImageNet values (which should theoretically improve performance). Finally, they are converted to Torch's Tensor in order to work properly with the model and the GPU optimization.

B. Model

For the model selection, I have opted for DeepLabv3 model as it is recognized to be well suited for semantic segmentation tasks. The model has three backbones available: ResNet50, ResNet101 and MobileNetv3. These are all available with their respective pre-trained weights. ResNet101 has a deeper model than ResNet50, therefore requiring additional training time and VRAM. With that in mind, I have opted for a ResNet50 backbone with it's associated pre-trained weights as the accuracy differences are fairly minimal while allowing to run more complex trainings.

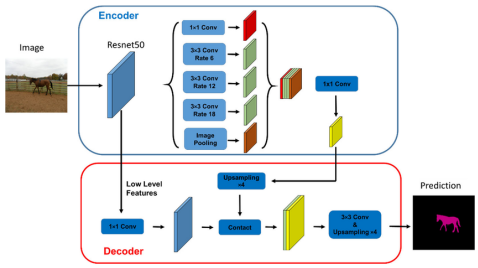

Fig. 3. DeepLabv3 with ResNet50 backbone architecture.

The configuration of the models did require some adjustments based on the amount of classes required for training and inference. Since our dataset has 20 classes to train on, the ResNet's convolutional layers had to be tweaked with these parameters:

```
model.classifier[4] = nn.Conv2d(256, 20,
kernel_size=1)
if model.aux_classifier:
    model.aux_classifier[4] =
    nn.Conv2d(256, 20, kernel_size=1)
```

III. Results & Discussions

A. Training

For training, I have tested on various setup and hyper-parameters in order to optimize the accuracy. The strategy was mostly aimed at optimizing the validation mIoU (mean Intersection over Union) as this would optimize the model on unseen Cityscape's data.

The prototyping and optimization for the training was done on a single H100 80GB GPU. This was to determine the boundaries of stability on my model and properly build the architecture. For the final training run, I used 8xA100 40GB GPUs as this would allow me to increase the input resolution.

The final training run was set up for **20 epochs**, had **height 768** and **width 1024**. The hyper-parameters were set up as followed:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
lr=0.0001)
```

We use a Cross-Entropy loss, and the Adam optimizer with a learning rate of 0.0001. On the parallel GPUs training, the batch size is of 24 with 8 workers. The training took approximately 20min, with the mIoU computed every epoch for the training and validation pass, giving me a good idea of the effectiveness of the training.

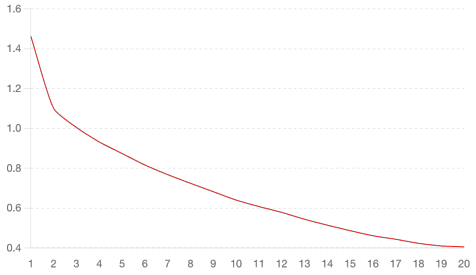

Fig. 4. Loss progression during training.

In the figure above, we can observe the loss curve (for the training data), displaying that model is training appropriately given the hyper-parameters.

As for the mIoU, it was evaluated during each epoch as a mean of estimating the accuracy progress on the training and validation pass.

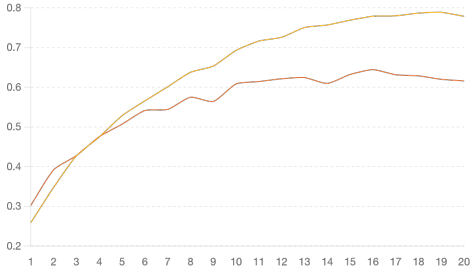

Fig. 5. mIoU progression on the training and validation pass. *Yellow: Training mIoU, Orange: Validation mIoU.*

In figure 5, we observe the validation mIoU flattening out at about the 12th epoch indicating the mIoU has likely reached

its best performance given the current setup. This gives us an approximate $\sim$**60.9% to 64.4%** mIoU (Range from epoch 12 to 20)(Avg. mIoU: 0.6255, Std. Dev.: 0.0096) on unseen Validation Data as a performance result.

B. Evaluation

After having trained the model, the state dictionary was saved as a checkpoint in order to re-evaluate the model. Because the model was trained under $nn.DataParallel$ mode, we need to re-initialize the model in that form even for single GPU inference. The resulting mIoU on all the validation data on the fine-tuned model is **61.6%** as our final result. This contrasts to the baseline mIoU which is **0.4%**. This demonstrate a major contrast, as the baseline model has no idea how to classify our classes as desired as seen in figure 6.

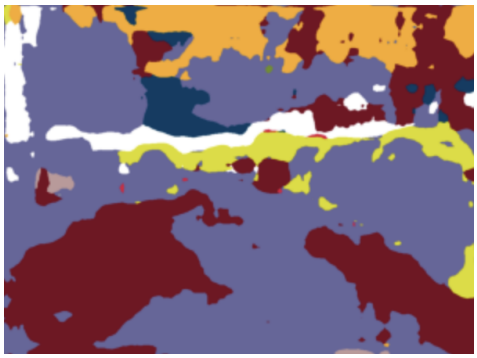

Fig. 6. Predictions using the pre-trained baseline model.

Fig. 7. Final predictions using our fine-tuned model.

The model has seen drastic improvements as the original model, despite having been pre-trained on the ImageNet dataset, was simply not able to infer how the different classes should've been classified.

C. Possible improvements

As observed during training, the model has seemingly reached a point in which the validation mIoU could not be improved much further. Given more time, it would've been interesting to implement further techniques such as additional data augmentation, or adaptable learning rates to try to break past the threshold of 62% mIoU.

Given more computing power, an approach such as 5-fold Cross Validation would've been ideal, but training times and instability with large models we're a constraining factor.

CONCLUSION

In this study, we have demonstrated the effectiveness of fine-tuning the DeepLabv3 model with a ResNet-50 backbone on the Cityscapes dataset. By leveraging the pre-trained weights and adjusting the model's architecture to accommodate our specific dataset, we achieved significant improvements in semantic segmentation accuracy. The fine-tuned model demonstrated a considerable performance boost with an average mIoU of 61.6%, compared to the baseline model's 0.4%. These results highlight the potential of model fine-tuning in improving the precision of domain-specific image analysis tasks. Future work could explore additional techniques to further improve segmentation performance.

REFERENCES

[1] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017.

[2] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020.

[3] T. maintainers and contributors, "Torchvision: Pytorch's computer vision library." https://github.com/pytorch/vision, 2016.