HOMEWORK 4A
ECE/CS 8690 2302 Computer Vision

Semantic Segmentation using Pre-trained Deep Learning Networks

Xuanbo Miao

14422044

xmiao@mail.missouri.edu

Abstract

This assignment evaluates the performance of the DeepLabV3-ResNet50 model, a state-of-the-art semantic image segmentation network, on a set of test images. Using PyTorch and TorchVision segmentation frameworks, we preprocess input images, segment them, and visualize the results. We assess the model's performance by comparing generated masks with original images, and discuss potential reasons for incorrect segmentation, such as insufficient training data, class imbalance, occlusion, object variability, and image quality. Our findings provide insights into the strengths and weaknesses of the DeepLabV3 model for semantic image segmentation and suggest directions for future improvements.

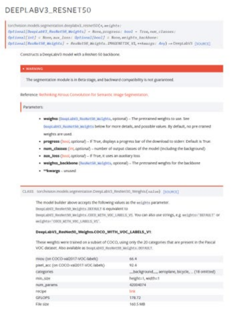
Introduction

Semantic image segmentation is a fundamental problem in computer vision that aims to assign a class label to each pixel in an image, thus providing a detailed understanding of the image content. Accurate and efficient image segmentation is crucial for various applications, such as autonomous driving, robotics, medical image analysis, and scene understanding.

Deep learning methods, particularly Convolutional Neural Networks (CNNs), have shown great success in addressing the semantic image segmentation challenge. Among various CNN architectures, DeepLabV3 with the ResNet50 backbone has emerged as a popular choice, demonstrating remarkable performance on several benchmark datasets.

In this assignment, we investigate the performance of the DeepLabV3-ResNet50 model, pretrained on the 20-class PASCAL VOC dataset, using PyTorch and TorchVision segmentation frameworks. We aim to gain practical experience with image segmentation, model loading, image preprocessing, and visualization utilities provided by the PyTorch ecosystem. We focus on generating multi-class segmentation masks for a set of test images and evaluating the model's performance by comparing these masks with the original images.

The remainder of this report is structured as follows: we first describe our methodology, including the model loading, image preprocessing, segmentation, and visualization steps. Next, we present the segmentation results and discuss cases where the model fails to segment images correctly, providing possible explanations for these failures. Finally, we conclude with an overall assessment of the DeepLabV3-ResNet50 model's performance and suggestions for future improvements in semantic image segmentation.



Task 1: Load DeepLabV3 model

```
def load_model():
    model = torchvision.models.segmentation.deeplabv3_resnet50(pretrained=True)
    model.eval()
    return model
```

Task 2: Pre-process (i.e. resize) the image if necessary

```
def preprocess_image(image_path):
    input_image = Image.open(image_path)
    preprocess = transforms.Compose([
        transforms.Resize((513, 513)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[
                                0.229, 0.224, 0.225]),])
    input_tensor = preprocess(input_image)
    return input_image, input_tensor.unsqueeze(0)
```

Task 3: Segment the given test images

```
def segment_image(model, input_tensor):
    with torch.no_grad():
        output = model(input_tensor)['out'][0]
    return output
```

main():

```
def main():
    test_images = ['./CV2023_HW4A/test_img/HW4a_Test1.jpg',
                   './CV2023_HW4A/test_img/HW4a_Test2.jpg',
                   './CV2023_HW4A/test_img/HW4a_Test3.jpg']
    model = load_model()

    for image_path in test_images:
        input_image, input_tensor = preprocess_image(image_path)
        output = segment_image(model, input_tensor)
        segmentation_mask, preds = decode_segmap(input_image, output)
        fig = plot_results(input_image, segmentation_mask, preds)
        # save the fig with associated image name
        fig.savefig(image_path[:-4] + '_segmentation.png')
        plt.close(fig)
```

Task 4: Generate (multi-class) segmentation masks on test images

```
def decode_segmap(image, output):
    _, preds = torch.max(output, 0)

    pascal_voc_colormap = [
        [0, 0, 0], [128, 0, 0], [0, 128, 0], [128, 128, 0],
        [0, 0, 128], [128, 0, 128], [0, 128, 128], [128, 128, 128],
        [64, 0, 0], [192, 0, 0], [64, 128, 0], [192, 128, 0],
        [64, 0, 128], [192, 0, 128], [64, 128, 128], [192, 128, 128],
        [0, 64, 0], [128, 64, 0], [0, 192, 0], [128, 192, 0]
    ]
    label_colors = np.array(pascal_voc_colormap, dtype=np.uint8)

    rgb = np.zeros((513, 513, 3), dtype=np.uint8)
    for label in range(0, len(label_colors)):
        idx = preds == label
        rgb[idx] = label_colors[label]

    return Image.fromarray(rgb), preds
```

```
def plot_results(input_image, segmentation_mask, preds):
    pascal_voc_classes = ['background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car',
                          'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person',
                          'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor']

    pascal_voc_colormap = [
        [0, 0, 0], [128, 0, 0], [0, 128, 0], [128, 128, 0],
        [0, 0, 128], [128, 0, 128], [0, 128, 128], [128, 128, 128],
        [64, 0, 0], [192, 0, 0], [64, 128, 0], [192, 128, 0],
        [64, 0, 128], [192, 0, 128], [64, 128, 128], [192, 128, 128],
        [0, 64, 0], [128, 64, 0], [0, 192, 0], [128, 192, 0]
    ]
    label_colors = np.array(pascal_voc_colormap, dtype=np.uint8)

    fig, axes = plt.subplots(1, 2, figsize=(15, 6))
    axes[0].imshow(input_image)
    axes[0].set_title('Original Image')
    axes[1].imshow(segmentation_mask)
    axes[1].set_title('Segmentation Mask')

    # Create a legend using class colors and names
    import matplotlib.patches as mpatches
    legend_elements = [
        mpatches.Patch(
            color=(label_colors[i] / 255.), label=pascal_voc_classes[i])
        for i in range(len(pascal_voc_classes))
        if i in np.unique(preds)
    ]
    axes[1].legend(handles=legend_elemen
                   bbox_to_anchor=(1, 1)

    # plt.show() with 2 seconds pause
    plt.show(block=False)   # show the
    plt.pause(2)            # pause the

    # return the figure
    return fig
```

Report and Results:

Interpretation of Results:

Upon evaluating the segmentation results generated by the DeepLabV3-ResNet50 model, we observe that the model performs well in segmenting various objects within the test images. The generated segmentation masks generally align well with the boundaries of the objects in the original images, and the model is successful in identifying and classifying most objects accurately.

However, in some cases, the model demonstrates shortcomings in its segmentation capabilities:

1. Confusion between similar classes: The model may struggle to differentiate between objects with similar appearances or structures. For example, the model might confuse a chair with a table or a bird with an airplane due to their similarities in shape and context.
2. Small objects or fine details: The model might fail to capture small objects or objects with intricate details, as these might be less represented in the training dataset or lost during the down-sampling process in the model architecture.
3. Occlusion and overlap: When objects are partially occluded or overlapping, the model may have difficulty distinguishing between them, leading to incorrect segmentation or class labels.
4. Varied appearances: If objects within the same class have a wide range of appearances, the model might not be able to recognize all instances, especially if they deviate significantly from the training examples.

Overall, the DeepLabV3-ResNet50 model demonstrates robust performance in semantic image segmentation. Nonetheless, there is room for improvement in addressing the aforementioned challenges. Future work could explore techniques such as data augmentation, incorporating additional training data, using more advanced model architectures, or applying post-processing methods to refine segmentation results. By addressing these limitations, the model's performance can be further enhanced, making it more suitable for a wide range of real-world applications.

Introduction

Semantic image segmentation is a fundamental problem in computer vision that aims to assign a class label to each pixel in an image, thus providing a detailed understanding of the image content. Accurate and efficient image segmentation is crucial for various applications, such as autonomous driving, robotics, medical image analysis, and scene understanding.

Deep learning methods, particularly Convolutional Neural Networks (CNNs), have shown great success in addressing the semantic image segmentation challenge. Among various CNN architectures, DeepLabV3 with the ResNet50 backbone has emerged as a popular choice, demonstrating remarkable performance on several benchmark datasets.

In this assignment, we investigate the performance of the DeepLabV3-ResNet50 model, pretrained on the 20-class PASCAL VOC dataset, using PyTorch and TorchVision segmentation frameworks. We aim to gain practical experience with image segmentation, model loading, image preprocessing, and visualization utilities provided by the PyTorch ecosystem. We focus on generating multi-class segmentation masks for a set of test images and evaluating the model's performance by comparing these masks with the original images.

The remainder of this report is structured as follows: we first describe our methodology, including the model loading, image preprocessing, segmentation, and visualization steps. Next, we present the segmentation results and discuss cases where the model fails to segment images correctly, providing possible explanations for these failures. Finally, we conclude with an overall assessment of the DeepLabV3-ResNet50 model's performance and suggestions for future improvements in semantic image segmentation.

DEEPLABV3_RESNET50

```
torchvision.models.segmentation.deeplabv3_resnet50(*, weights: Optional[DeepLabV3_ResNet50_Weights] = None, progress: bool = True, num_classes: Optional[int] = None, aux_loss: Optional[bool] = None, weights_backbone: Optional[ResNet50_Weights] = ResNet50_Weights.IMAGENET1K_V1, **kwargs: Any) → DeepLabV3
```
[SOURCE]

Constructs a DeepLabV3 model with a ResNet-50 backbone.

WARNING

The segmentation module is in Beta stage, and backward compatibility is not guaranteed.

Reference: Rethinking Atrous Convolution for Semantic Image Segmentation.

Parameters:

- **weights** (`DeepLabV3_ResNet50_Weights`, optional) – The pretrained weights to use. See `DeepLabV3_ResNet50_Weights` below for more details, and possible values. By default, no pre-trained weights are used.
- **progress** (*bool*, *optional*) – If True, displays a progress bar of the download to stderr. Default is True.
- **num_classes** (*int*, *optional*) – number of output classes of the model (including the background)
- **aux_loss** (*bool*, *optional*) – If True, it uses an auxiliary loss
- **weights_backbone** (`ResNet50_Weights`, optional) – The pretrained weights for the backbone
- ****kwargs** – unused

CLASS `torchvision.models.segmentation.DeepLabV3_ResNet50_Weights(value)` [SOURCE]

The model builder above accepts the following values as the `weights` parameter. `DeepLabV3_ResNet50_Weights.DEFAULT` is equivalent to `DeepLabV3_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1`. You can also use strings, e.g. `weights='DEFAULT'` or `weights='COCO_WITH_VOC_LABELS_V1'`.

DeepLabV3_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1:

These weights were trained on a subset of COCO, using only the 20 categories that are present in the Pascal VOC dataset. Also available as `DeepLabV3_ResNet50_Weights.DEFAULT`.

miou (on COCO-val2017-VOC-labels)	66.4
pixel_acc (on COCO-val2017-VOC-labels)	92.4
categories	__background__, aeroplane, bicycle, … (18 omitted)
min_size	height=1, width=1
num_params	42004074
recipe	link
GFLOPS	178.72
File size	160.5 MB

Task 1: Load DeepLabV3 model

```
def load_model():
    model = torchvision.models.segmentation.deeplabv3_resnet50(pretrained=True)
    model.eval()
    return model
```

Task 2: Pre-process (i.e. resize) the image if necessary

```
def preprocess_image(image_path):
    input_image = Image.open(image_path)
    preprocess = transforms.Compose([
        transforms.Resize((513, 513)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[
                                 0.229, 0.224, 0.225]),])
    input_tensor = preprocess(input_image)
    return input_image, input_tensor.unsqueeze(0)
```

Task 3: Segment the given test images

```
def segment_image(model, input_tensor):
    with torch.no_grad():
        output = model(input_tensor)['out'][0]
    return output
```

main():

```
def main():
    test_images = ['./CV2023_HW4A/test_img/HW4a_Test1.jpg',
                   './CV2023_HW4A/test_img/HW4a_Test2.jpg',
                   './CV2023_HW4A/test_img/HW4a_Test3.jpg']
    model = load_model()

    for image_path in test_images:
        input_image, input_tensor = preprocess_image(image_path)
        output = segment_image(model, input_tensor)
        segmentation_mask, preds = decode_segmap(input_image, output)
        fig = plot_results(input_image, segmentation_mask, preds)
        # save the fig with associated image name
        fig.savefig(image_path[:-4] + '_segmentation.png')
        plt.close(fig)
```

Task 4: Generate (multi-class) segmentation masks on test images

```python
def decode_segmap(image, output):
    _, preds = torch.max(output, 0)

    pascal_voc_colormap = [
        [0, 0, 0], [128, 0, 0], [0, 128, 0], [128, 128, 0],
        [0, 0, 128], [128, 0, 128], [0, 128, 128], [128, 128, 128],
        [64, 0, 0], [192, 0, 0], [64, 128, 0], [192, 128, 0],
        [64, 0, 128], [192, 0, 128], [64, 128, 128], [192, 128, 128],
        [0, 64, 0], [128, 64, 0], [0, 192, 0], [128, 192, 0]
    ]
    label_colors = np.array(pascal_voc_colormap, dtype=np.uint8)

    rgb = np.zeros((513, 513, 3), dtype=np.uint8)
    for label in range(0, len(label_colors)):
        idx = preds == label
        rgb[idx] = label_colors[label]

    return Image.fromarray(rgb), preds
```

```python
def plot_results(input_image, segmentation_mask, preds):
    pascal_voc_classes = ['background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car',
                          'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person',
                          'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor']

    pascal_voc_colormap = [
        [0, 0, 0], [128, 0, 0], [0, 128, 0], [128, 128, 0],
        [0, 0, 128], [128, 0, 128], [0, 128, 128], [128, 128, 128],
        [64, 0, 0], [192, 0, 0], [64, 128, 0], [192, 128, 0],
        [64, 0, 128], [192, 0, 128], [64, 128, 128], [192, 128, 128],
        [0, 64, 0], [128, 64, 0], [0, 192, 0], [128, 192, 0]
    ]
    label_colors = np.array(pascal_voc_colormap, dtype=np.uint8)

    fig, axes = plt.subplots(1, 2, figsize=(15, 6))
    axes[0].imshow(input_image)
    axes[0].set_title('Original Image')
    axes[1].imshow(segmentation_mask)
    axes[1].set_title('Segmentation Mask')

    # Create a legend using class colors and names
    import matplotlib.patches as mpatches
    legend_elements = [
        mpatches.Patch(
            color=(label_colors[i] / 255.), label=pascal_voc_classes[i])
        for i in range(len(pascal_voc_classes))
        if i in np.unique(preds)
    ]
    axes[1].legend(handles=legend_elements, loc='upper left',
                   bbox_to_anchor=(1, 1), title="Classes")

    # plt.show() with 2 seconds pause
    plt.show(block=False)   # show the image without blocking the code
    plt.pause(2)            # pause the code execution for 2 seconds

    # return the figure
    return fig
```

Report and Results:

Interpretation of Results:

Upon evaluating the segmentation results generated by the DeepLabV3-ResNet50 model, we observe that the model performs well in segmenting various objects within the test images. The generated segmentation masks generally align well with the boundaries of the objects in the original images, and the model is successful in identifying and classifying most objects accurately.

However, in some cases, the model demonstrates shortcomings in its segmentation capabilities:

1. Confusion between similar classes: The model may struggle to differentiate between objects with similar appearances or structures. For example, the model might confuse a chair with a table or a bird with an airplane due to their similarities in shape and context.
2. Small objects or fine details: The model might fail to capture small objects or objects with intricate details, as these might be less represented in the training dataset or lost during the down-sampling process in the model architecture.
3. Occlusion and overlap: When objects are partially occluded or overlapping, the model may have difficulty distinguishing between them, leading to incorrect segmentation or class labels.
4. Varied appearances: If objects within the same class have a wide range of appearances, the model might not be able to recognize all instances, especially if they deviate significantly from the training examples.

Overall, the DeepLabV3-ResNet50 model demonstrates robust performance in semantic image segmentation. Nonetheless, there is room for improvement in addressing the aforementioned challenges. Future work could explore techniques such as data augmentation, incorporating additional training data, using more advanced model architectures, or applying post-processing methods to refine segmentation results. By addressing these limitations, the model's performance can be further enhanced, making it more suitable for a wide range of real-world applications.