Task 2: Train an Image Segmentation Model

https://github.com/mfarhadhussain/mask_generation_segmentation.git

Md Farhad Hussain

1 Overview

This task involves training an image segmentation model using PyTorch. The objective is to segment objects in images using the masks prepared in Task 1. The model can be trained for binary or multi-class segmentation based on the input dataset.

2 Model Architecture

The model is based on an encoder-decoder (U-Net) architecture with the following key features:

- **Residual Blocks:** Each block uses two convolutional layers with batch normalization and a SiLU activation. Shortcut connections are added to improve gradient flow and overall convergence.
- **Self-Attention Block:** Integrated into the bottleneck, this block computes attention scores across feature maps, weighting them to capture long-range dependencies. The attention is computed using a scaled dot-product approach.
- **Skip Connections:** The U-Net structure utilizes skip connections between the encoder and decoder layers, which help to preserve spatial details.

3 Loss Functions and Optimization

Several loss functions were developed and experimented with:

- **Simple Loss:** A basic cross-entropy loss function.
- **Dice Cross-Entropy Loss:** Combines cross-entropy and Dice loss to balance region-level performance.
- **Weighted Cross-Entropy Loss:** Uses class weights computed from mask statistics to address class imbalance.

training resulted reported here for the `SimpleLoss`. The model weights are optimized using the Adam optimizer with an initial learning rate of 1×10^{-4}, along with a cosine annealing learning rate scheduler.

4 Dataset and Preprocessing

The dataset is based on COCO segmentation images and corresponding masks. The `CocoSegmentationDataset` class performs the following:

- Resizing of images and masks to a fixed resolution (256×256).
- Converting images to tensors.
- Automatically generating label mapping by scanning mask pixel values.

To mitigate class imbalance, sample weights are computed based on the inverse frequency of classes in the dataset. A `WeightedRandomSample` is used within the DataLoader to ensure balanced training.

5 Training and Validation Procedure

5.1 Training

The training loop is organized into epochs with the following steps:

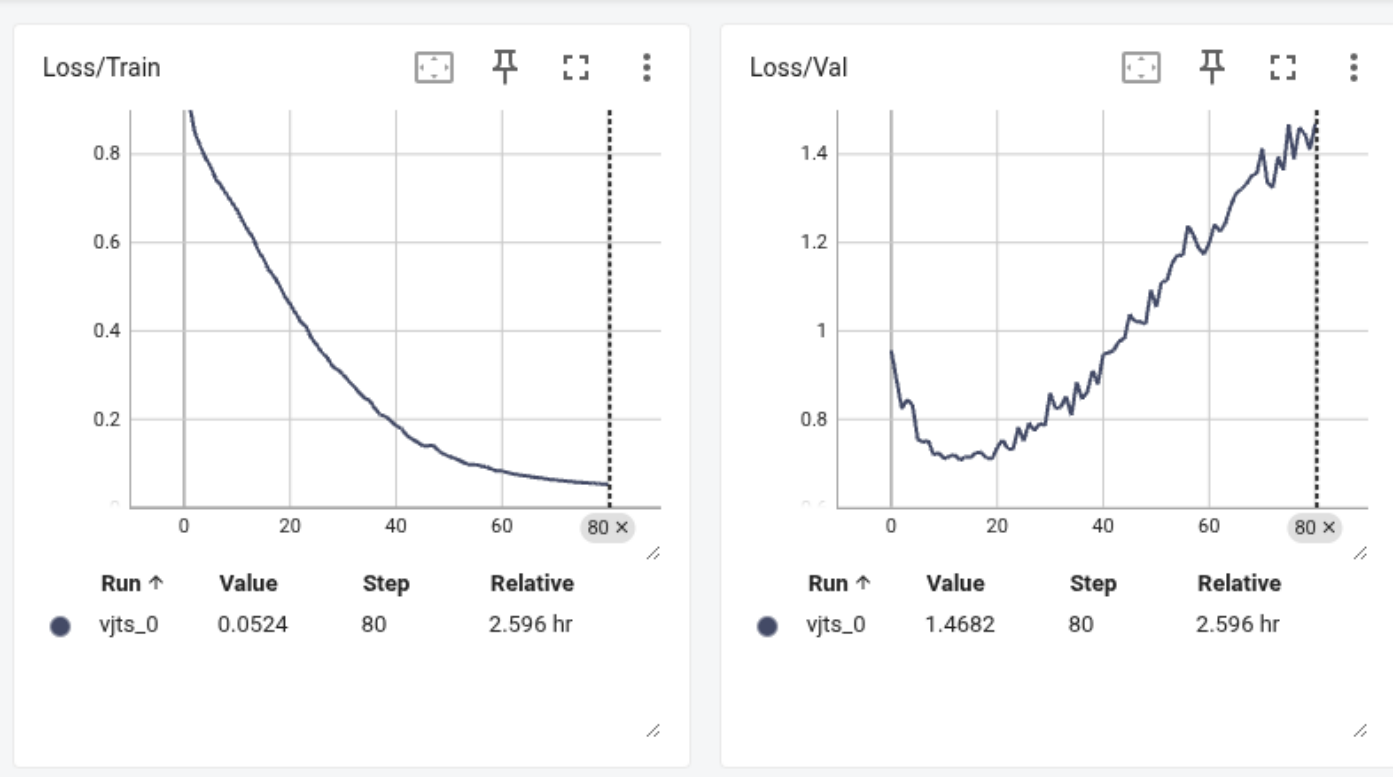
1. Forward propagation to compute segmentation predictions.
2. Loss computation using the chosen loss function.
3. Backward propagation and optimization.
4. Evaluation on a validation set using metrics: Intersection over Union (IoU), Dice coefficient, and pixel accuracy.

Model checkpoints are saved every 10 epochs along with sample predictions saved as images for visual inspection. Additionally, training metrics are logged using TensorBoard for real-time monitoring.

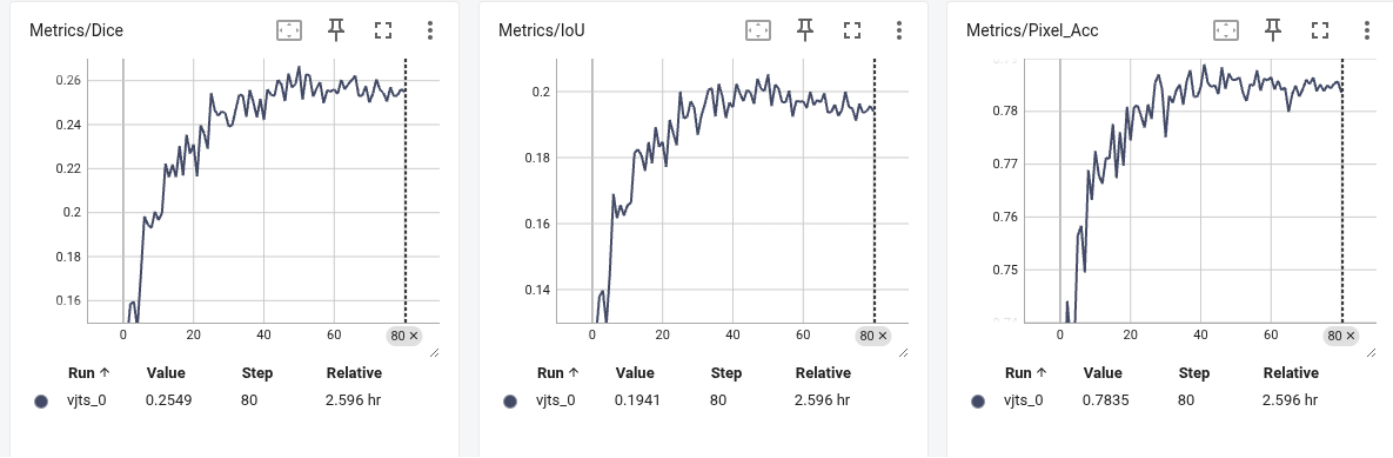
5.2 Evaluation Metrics

The performance of the segmentation model is evaluated using:

- **IoU:** Measures the overlap between the predicted segmentation and ground truth.
- **Dice Coefficient:** Assesses the similarity between the predicted mask and the ground truth.
- **Pixel Accuracy:** The ratio of correctly predicted pixels to the total pixels.



Figur 1: Loss curve of training and validation



Figur 2: Dice, IoU and Pixcel accuracy on Validation dataset

5.3 Observation

The training loss decreases as expected; however, the validation loss begins to increase after approximately 20 epochs, indicating that the model is overfitting the training data. This overfitting is likely caused by the relatively small size of the training dataset (only 5000 images) and the validation dataset (2000 images).

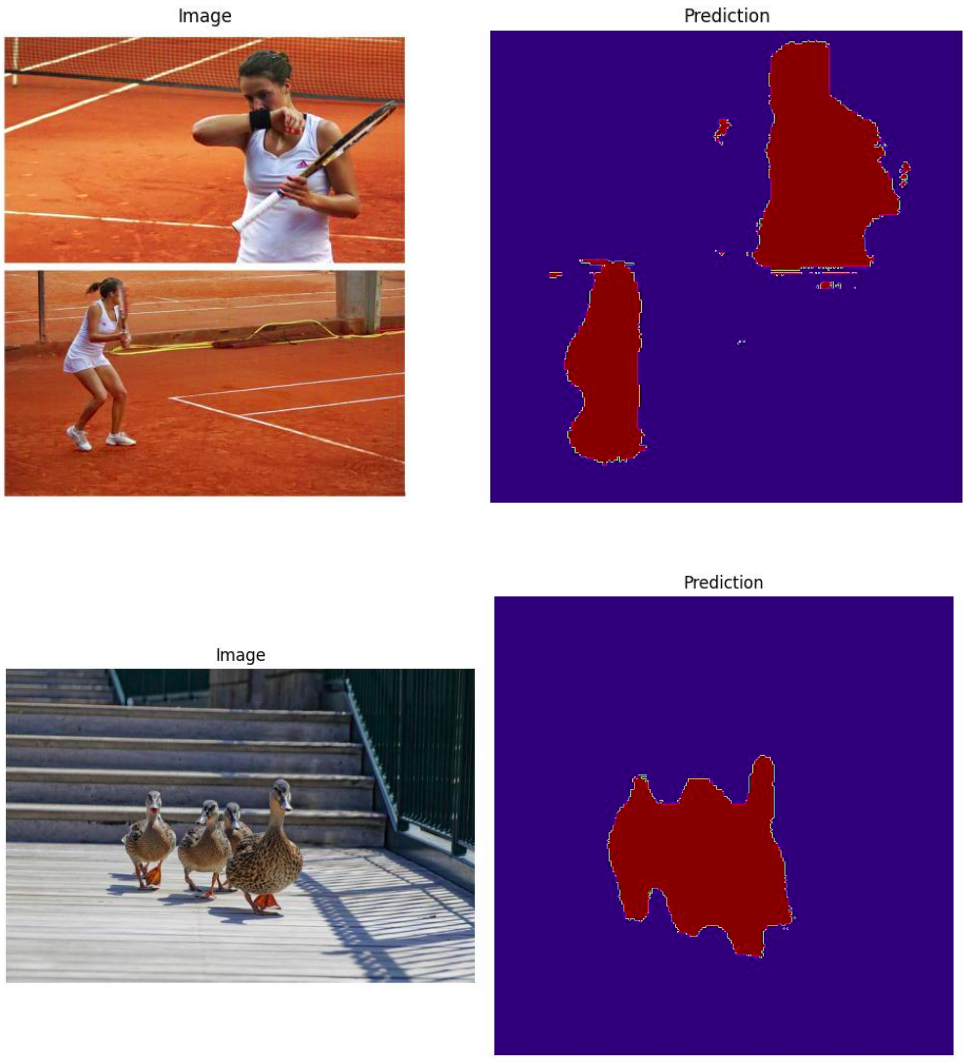
To mitigate overfitting, several approaches can be considered:

- **Increasing the dataset size**: Expanding the training dataset would help the model generalize better and reduce the overfitting effect. This can be achieved by collecting more data or using data augmentation techniques.
- **Data Augmentation**: Applying transformations such as horizontal flipping, rotational flips, and random brightness adjustments during training can help the model become more robust by introducing variability in the data.
- **Regularization techniques**: Introducing a dropout layer during model training can help prevent overfitting by randomly disabling certain neurons during training, forcing the model to learn more generalizable features.

By applying these methods, the model's generalization ability is expected to improve, reducing overfitting on the training set and enhancing performance on unseen data.

6 Results

The result is produced here on using the 20th epoch, on unseen data.



Figur 3: Segmentation result on unseen images.

7 Conclusion

This task demonstrates an end-to-end training pipeline for image segmentation using PyTorch. and to do the inference on numbber of images by passing the image directory path to inference.py script.