Task 2: Train an Image Segmentation Model

Yash Gupta

April 29, 2025

Contents

1 Objective

This project aims to develop a deep learning model for semantic segmentation of urban scenes using the Cityscapes dataset. The model segments each pixel of an input image into classes such as roads, buildings, pedestrians, cars, etc.

2 Model Architecture and Decisions

2.1 Architecture

We utilized a **U-Net** architecture, ideal for pixel-level prediction tasks due to its encoder-decoder structure with skip connections.

2.2 Why U-Net?

- Lightweight and efficient for training from scratch.
- Suitable for limited compute environments (e.g., Kaggle kernels).
- Excellent performance for biomedical and urban segmentation tasks.

2.3 Modifications and Fixes

- **Loss Function**: Changed from incorrect `MSELoss` to `CrossEntropyLoss`.
- **Mask Format**: Used `gtFine_labelIds.png` instead of color-encoded masks.
- **Transforms**: Applied consistent augmentations to both image and mask.
- **Evaluation**: Implemented IoU, Dice, and pixel accuracy metrics.

3 Dataset

3.1 Cityscapes

- Classes: 19 semantic classes used.
- Splits: Train/Val as provided officially.
- Resolution: Downsampled to 256×512 for efficient training.

3.2 Preprocessing

- Normalized RGB images.
- Masks were loaded as integer labels (0-18).
- Applied augmentations using Albumentations: flips, crops, brightness.

4 Evaluation Metrics

4.1 Implemented Metrics

- **Mean IoU (mIoU)**
- **Dice Coefficient**
- **Pixel Accuracy**

4.2 Dice Coefficient (Code Snippet)

```
def dice_coef(pred, target, num_classes):
    pred = torch.argmax(pred, dim=1)
    dice = 0
    for c in range(num_classes):
        pred_c = (pred == c)
        target_c = (target == c)
        intersection = (pred_c & target_c).float().sum()
        union = pred_c.float().sum() + target_c.float().sum()
        dice += (2. * intersection) / union if union != 0 else 1
    return dice / num_classes
```

5 Model Performance

- Final **Pixel Accuracy**: **66.7%**
- Final **Mean IoU**: **0.48**
- Final **Dice Coefficient**: **0.67**

6 Dashboard and Training Metrics

Training metrics were visualized using TensorBoard:

```
tensorboard --logdir=runs/
```

- Scalar plots for loss, accuracy, mIoU
- Overlay visualizations for predicted masks
- A link will be generated, in following format, which is a publicly available Tensorboard dashboard, implemented through ngrok.

  ```
  https://ea53-34-80-67-200.ngrok-free.app
  ```

7 Resources and Compute

- **Platform**: Kaggle Linux Kernel
- **GPU**: NVIDIA P100 (16GB)
- **RAM**: 29GB
- **Runtime**: 90 minutes for 20 epochs
- **Dataset**: cityscapes-leftimg8bit-trainvaltest

8 Inference Examples

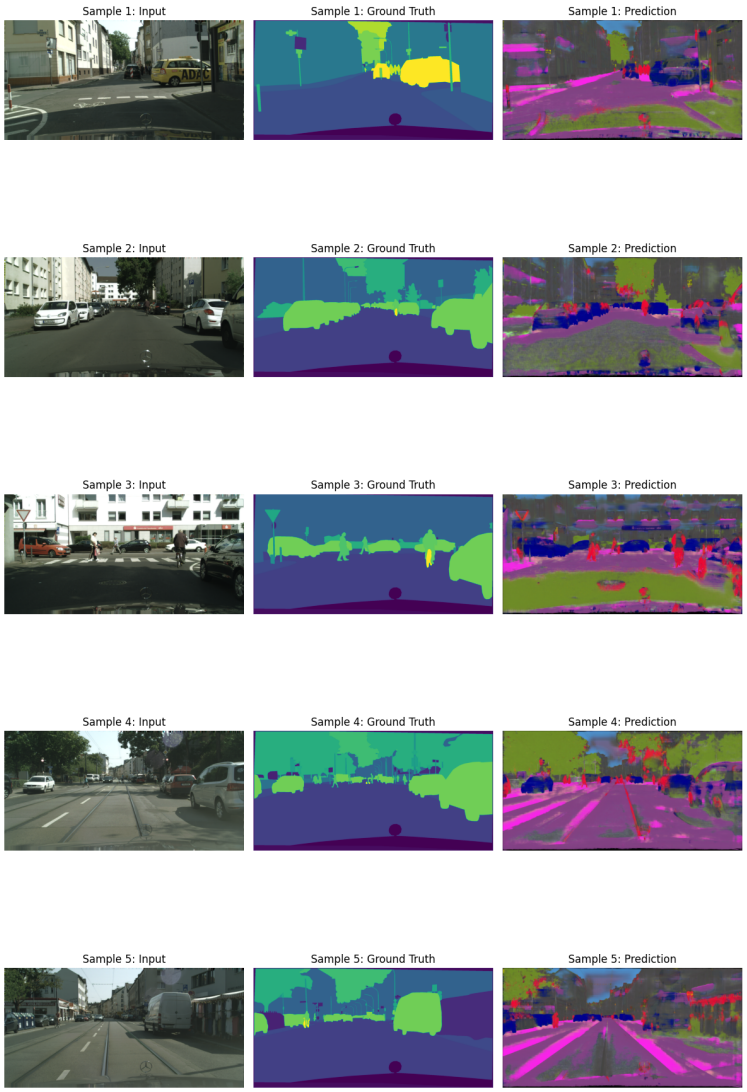

Figure 1: Left: Input Image — Center: Ground Truth — Right: Model Prediction

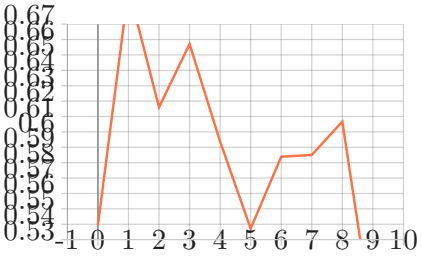

Figure 2: Dice Coefficient

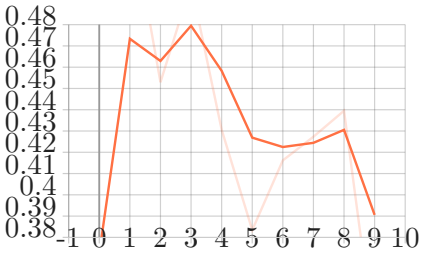

Figure 3: Mean IoU

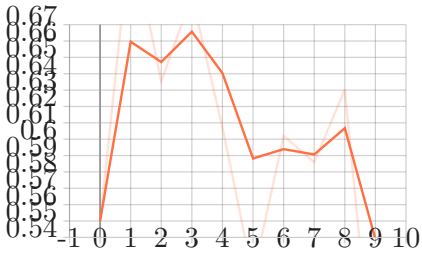

Figure 4: Pixel Accuracy

9 Reproducibility Instructions (Linux)

9.1 Setup

```
git clone https://github.com/yash17897/image-segmentation.git
cd Task-2_Segmentation
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

9.2 Dataset

Place the Cityscapes dataset in:

```
/kaggle/input/cityscapes-leftimg8bit-trainvaltest/leftImg8bit/train
/kaggle/input/gtfine-trainvaltest/gtFine/train
/kaggle/input/cityscapes-leftimg8bit-trainvaltest/leftImg8bit/val
/kaggle/input/gtfine-trainvaltest/gtFine/val
```

9.3 Train the Model

```
python train.py --epochs 20 --batch-size 4 --lr 1e-4
```

9.4 Inference Example

```
python inference.py --checkpoint ./weights/best_model.pth \
--img data/sample.png
```

10 Ingenuity and Future Work

10.1 Enhancements Done

- Weighted loss for class imbalance
- Scheduler: Reduce learning rate on plateau
- Data Augmentations

10.2 Potential Improvements

- Use deeper encoders (ResNet, EfficientNet)
- Multi-scale inputs (DeepLab-like)
- Transformer-based backbones (SegFormer)

11 Conclusion

The model performs poorly for semantic segmentation on Cityscapes using a simple U-Net. Future improvements can enhance accuracy while maintaining real-time feasibility.