Name: Avar Muhammed Yahya, Praneshraj Tiruppur Nagarajan Dhyaneswar
Neptun Code: Y8OIJ1, AOMTO9
Subject: Advanced Analytics Lab

Urban Scene Understanding with Deep Learning

We present a project focused on urban scene understanding using deep learning techniques, where we applied boundary detection on the Cityscapes dataset.

Introduction

This project falls under the field of Computer Vision, with the goal of helping machines better understand city environments. For self-driving cars to move safely and make smart choices, they need to clearly recognize things like roads, people, vehicles, and traffic signs. To do this, we use techniques like boundary and object detection. We trained and evaluated our models using the popular Cityscapes dataset, which provides detailed images of urban scenes.

Boundary Detection

Also known as edge detection, it focuses on identifying the demarcation lines between different objects or semantic classes in an image. It is important for refining segmentation masks, understanding the object shapes, and providing comprehensive contours that help in navigation and interaction with the environment. In simple terms, precise boundary detection is essential for tasks like path planning and avoiding collisions by accurately delineating the edges of obstacles and the drivable path.

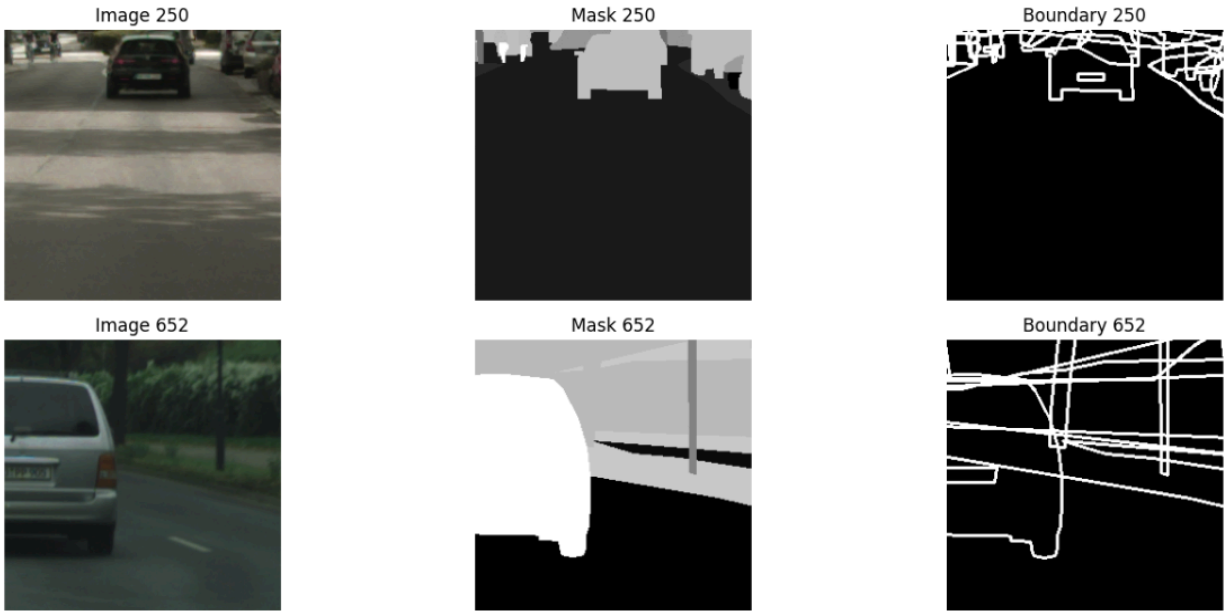
Boundary detection offers pixel-level detail about the edges of objects and semantic regions.

Data

The primary dataset used for this project is the **Cityscapes Dataset**. This dataset is widely recognized for urban scene understanding tasks due to its detailed annotations of urban street scenes.It contains 5,000 finely annotated images with detailed pixel-level semantic, instance, and panoramic segmentation, along with an additional 20,000 coarsely annotated images. For this project, a subset of **2,975** images was used for training and evaluation, comprising a combination of fine and coarse annotations to balance precision and dataset size.

Each sample in the dataset includes:

1. **leftImg8bit** images: These are high-resolution RGB images serving as the primary input for model training.
2. **polygons.json**: This file contains polygon coordinates that define the exact boundaries of object instances in the scene. A sample is shown below:



Preprocessing

The data preprocessing steps, as referenced in the Submission_Data_Preprocessing_and_Analytics_13052205.ipynb notebook, are critical for preparing the raw data for model training. The project initially used images of resolution 1024 × 1024, to closely preserve the original image details. However, this significantly overloaded the training script, causing excessive memory usage and prolonged training times. The resolution was scaled down to **256 × 256** to reduce computational overhead and allow subpar spatial detail for model learning.

The preprocessing pipeline was constructed using the **Albumentations** library, which provides fast and adaptable image transformations. This ensures that the model is exposed to a wide variety of scenarios. The steps differ slightly for training and validation.

For training dataset:

1. **RandomBrightnessContrast** (p=0.02): Slightly alters image brightness and contrast to simulate varying lighting conditions.
2. **GaussianBlur** (p=0.01, blur_limit=(3, 7)): To soften edges and simulate low-focus situations.
3. **ShiftScaleRotate** (p=0.03): Applied random shift (max 6.25%), scaling (max ±20%), and rotation (max ±15°) to simulate positional and geometric variations.
4. **Normalize** (mean = (0.485, 0.456, 0.406), std = (0.229, 0.224, 0.225)): Standardizes image pixel values.

For the validation dataset, only normalisation was performed.

Analytics Observed

Pixel Intensity Distribution

Crucial for preprocessing and thresholding in edge detection. Reveals image contrast and informs illumination invariance strategies.The chart indicates that the dataset has a significant number of pixels with mid-range intensity values, with a noticeable presence of both slightly darker and slightly brighter regions.

Normalized Contour Length

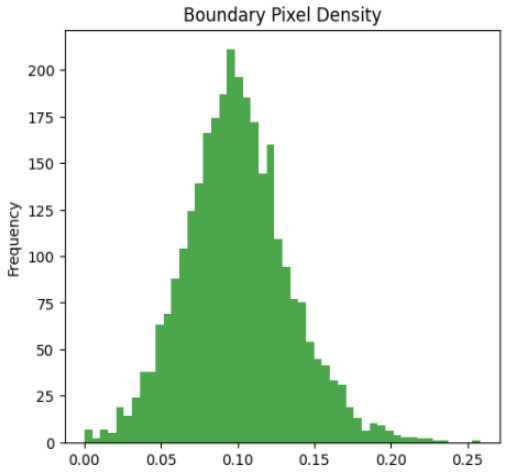
Measures boundary complexity. Peak around 0.03 suggests most images have relatively short boundaries relative to their size.

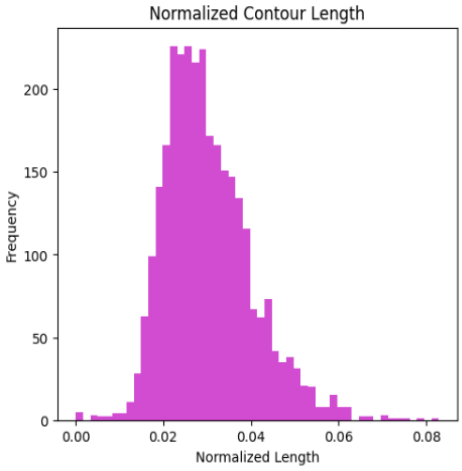
Edge Strength Distribution

Analyzes edge magnitudes, essential for threshold selection, noise reduction, and assessing edge detection quality. The chart shows that most intensity changes in the images are subtle, and strong, distinct edges are relatively rare.

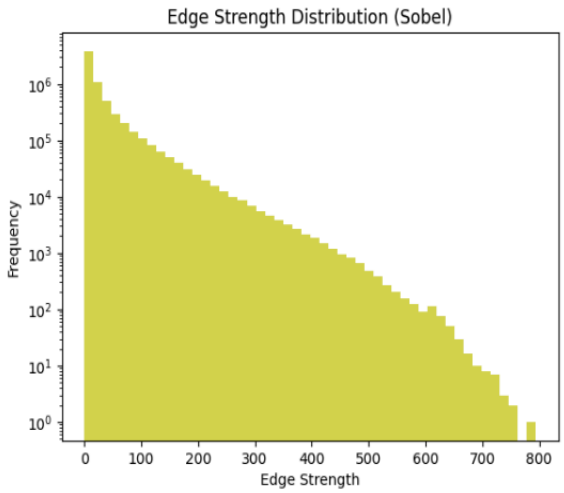
Boundary Pixel Density Distribution

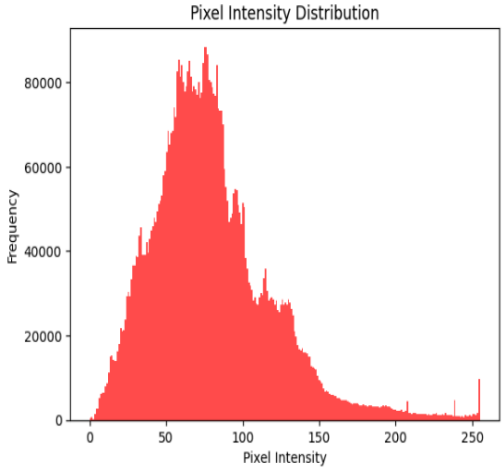
Quantifies the "edginess" of images, indicating scene complexity and guiding model selection. suggests that approximately 10% of the pixels in a typical image are part of an object boundary.





:





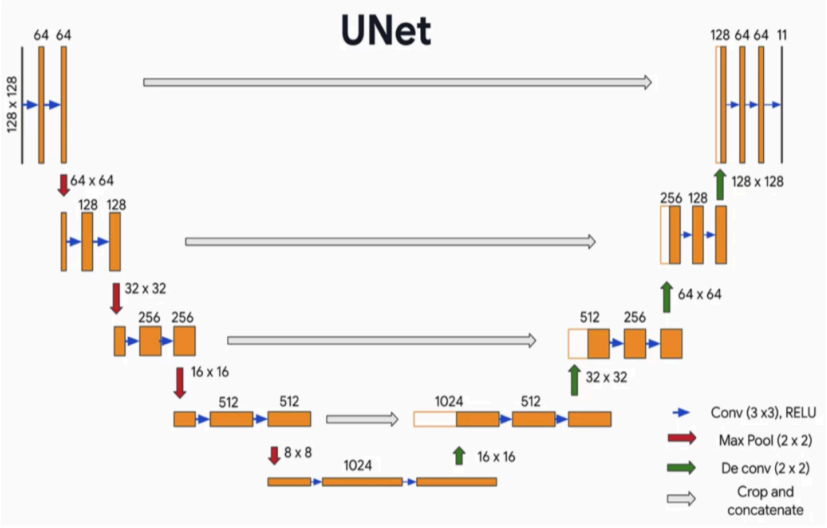
U-Net Model for Boundary Detection

U-Net is a convolutional neural network architecture comprising of two main parts:

1. **Encoder** (contracting path): Captures contextual features by progressively downsampling the input using convolution and max-pooling layers
2. **Decoder** (expanding path): Restores spatial resolution through upsampling and convolution. **Skip connections** link encoder and decoder layers at corresponding levels to retain fine spatial details.

This fusion of deep semantic features with shallow spatial cues enables U-Net to produce accurate segmentation and boundary masks, even in complex visual scenes.

Note: The figure below is for illustration purposes. In our model, the bottleneck occurs at the 512-channel layer



Observed U-Net Model

The U-Net model observed in the Submission_Cityscapes_UNET_13052025.ipynb notebook is tailored for the boundary detection task on the Cityscapes dataset. The implementation is built using the **PyTorch** deep learning framework.

Model Definition

The U-NET architecture was defined using PyTorch's torch.nn module, including the convolutional layers, pooling/upsampling operations, and skip connections.

1. Encoder has:
    a. 4 sequential blocks
    b. Each block (enc1 to enc4) comprises two **3×3** convolutional layers, then **Pooling.** (3 -> 64; 64-> 128; 128 - > 256; 256 ->512 (Bottleneck)
    c. Followed by **Batch Normalization** and **ReLU activation**.
    d. Downsampling performed after each block (except the last) using 2×2 Max Pooling with a stride of 2.
2. Decoder has:
    a. 3 upsampling and convolutional blocks (dec3 to dec1).
    b. The feature map is first upsampled using 2×2 bilinear interpolation to double its spatial dimensions.
    c. It is then concatenated with the corresponding feature map from the encoder through skip connections.
    d. This combined map passes through two 3×3 convolutional layers, each followed by Batch Normalization and ReLU activation.
    e. The number of channels is reduced by half at each step:
        i. dec3: (512 + 256) → 256 channels
        ii. dec2: (256 + 128) → 128 channels
        iii. dec1: (128 + 64) → 64 channels
3. Output Layer:
    a. The final layer is a 1×1 convolutional layer that maps the 64 feature channels from the last decoder block to a single output channel (out_channels=1). This output represents the raw logits for each pixel, indicating the likelihood of a boundary.

Training Parameters

The UNet model was trained using the following key parameters:

- **Optimizer:** Adam (α=1e−3)
- **Loss Function:** Binary Cross-Entropy with Logits Loss (nn.BCEWithLogitsLoss())
- **Batch Size:** 8 (effective batch size of 32 due to gradient accumulation)
- **Accumulation Steps:** 4
- **Number of Epochs per Chunk:** 40

- **Learning Rate:** 1×10−3
- **Data Augmentation:** Basic ToTensor() and Normalize() (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225] for images)
- **Mixed Precision Training:** Enabled on CUDA devices using torch.cuda.amp.GradScaler.
- **Chunked Training:** The dataset was processed in chunks of 1500 images to manage memory constraints.

The implementation was optimized for memory constraints due to the limitations of the training environment (Google Colab with limited RAM).

Evaluation

To evaluate boundary detection performance, we used three key metrics

1. **Evaluation Loss**

   Measures the difference between the predicted boundary mask and the actual ground truth mask. A lower loss indicates better predictions. **Binary Cross-Entropy** is used (since the output edge is denoted by 1 and rest 0).
   Note: *Please refer the script to assess the decreasing loss trend over epochs*

$$IoU = \frac{TP}{(TP + FP + FN)}$$

2. **Intersection over Union (IoU)**
   IoU measures how much the predicted and ground truth boundaries overlap.

3. **Dice Score (Dice Similarity Coefficient)**
   The Dice Score measures overlap but gives more importance to correctly predicted boundaries. It's especially useful when boundary regions are small compared to the background.

$$Dice = \frac{2 \times TP}{(TP + FP) + (TP + FN)}$$

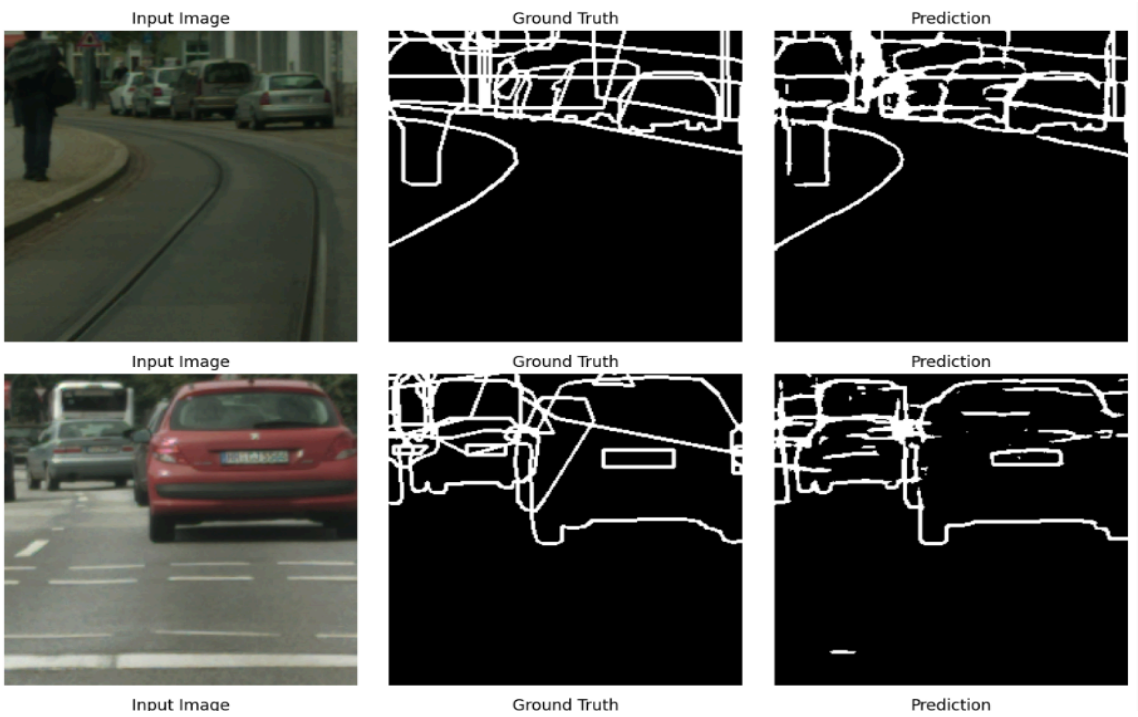
Equation 2. Dice similarity coefficient in terms of TP, FP and FN

For the boundary detection task, the model achieved

1. An average IOU score of **0.3016**
2. A Mean Dice Coefficient of **0.4809**
3. Training process concluded with a final training loss of **0.0948**.

These metrics indicate moderate overlap and similarity between predicted and ground truth boundaries, with mIoU suggesting challenges in precise spatial alignment despite the relatively low final training loss.

Visualisation



Note: The third column is the prediction of edges on eval dataset

Justification and Future Scope

The observed performance, despite a final training loss of 0.0948, subpar IOU and dice scores, is significantly influenced by the computational constraints of the Colab environment. Limited RAM necessitated a smaller batch size, potentially reduced input image resolution, and imposed architectural capacity trade-offs in our implementation. The discrepancy between the low training loss and the moderate evaluation metrics suggests potential overfitting to the training data within the constrained environment.

However, our UNet implementation, despite the computational constraints of the Colab environment, has successfully established a functional pipeline for boundary detection on the challenging Cityscapes dataset. Future research to enhance accuracy should prioritize scaling training infrastructure with greater GPU resources, higher resolutions of input training data, and more complex models beyond standard UNet or leveraging transfer learning from pre-trained models.

References

UNET

- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015* (pp. 234-241). Springer, Cham.
- Semantic segmentation of urban environments: Leveraging U-Net deep learning model for cityscape image analysis

Dataset

- The Cityscapes Dataset for Semantic Urban Scene Understanding

Scripts

- Submission_Data_Preprocessing_and_Analytics_13052205
- Submission_Cityscapes_UNET_13052025