VOC Segmentation App with Custom UNet

A semantic segmentation system that classifies every pixel in an image using a custom UNet model and presents the results in a visual app interface for real-time exploration.

Overview.

This project tackles the challenge of multi-class semantic segmentation on the Pascal VOC 2012 dataset. It features a custom UNet-like architecture built with ConvTranspose2d layers for upsampling and batch normalization to improve convergence. The model is trained and tracked using PyTorch Lightning and Weights & Biases (W&B), and wrapped inside a visual app for viewing predictions interactively.

Objective.

Build a UNet-style semantic segmentation model from scratch
Support all 21 VOC classes with pixel-wise accuracy
Normalize mask colors and log visual results live
Wrap the model in a user-friendly tool for visual testing

Tools.

- **Framework**: PyTorch Lightning
- **Model**: Custom UNet using Conv2d, ConvTranspose2d
- **Logging**: Weights & Biases
- **UI**: Matplotlib-based app for prediction browsing
- **Dataset**: Pascal VOC 2012
- **Visualization**: Matplotlib, PIL, custom colormaps

Process.

- Parse and convert RGB mask labels into integer masks
- Preprocess images and masks with resizing and normalization
- Build a UNet architecture with encoder-bottleneck-decoder
- Implement training, validation, and metric logging
- Log input, mask, and prediction side-by-side
- Use ModelCheckpoint to save top 3 models based on validation IoU
- Display batches as grids using Matplotlib within the app

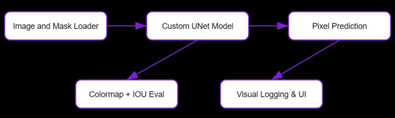
Core Features.

- **Custom UNet Architecture** using ConvTranspose2d for decoder upsampling
- **Multi-class segmentation** over 21 object classes from VOC 2012
- **Pixel-level prediction** with colormap visualizations
- **W&B Integration** for live logging and visualization
- **LightningModule + Callbacks** for clean, efficient training
- **App-based interface** to browse predictions, images, and masks
- **Top-K checkpointing** based on validation IoU

Challenges

- RGB mask decoding to class IDs was tricky due to non-unique colors
- Managing memory with large batch sizes and 21-channel outputs
- Balancing performance with real-time logging during training
- Creating a clean UI while maintaining GPU-based predictions

Flowchart



Results, Learnings, and Impact

- Learned to manipulate and normalize segmentation masks at pixel level
- Implemented a full training + validation pipeline in Lightning
- Discovered the power of batch normalization in stabilizing training
- Explored Weights & Biases for monitoring both metrics and visual samples
- Learned how to build apps that use trained models for image inspection
- Achieved **stable training and validation IoU** above baseline
- Consistently segmented foreground objects like people, dogs, and bikes
- Visual predictions were interpretable and tracked with live logging
- App was able to display 3-panel image comparison (input, mask, prediction) at each epoch