VJ Assignment Report

Aditya Laxman Kamble (M.Tech AI)

Task 1: Dataset Preparation using Python

Dataset Preparation

The dataset consists of input images of cars and their corresponding annotation files. Each annotation file contains polygon definitions for different class labels corresponding to different parts of a car, which are used to generate pixel-wise segmentation masks. The dataset preparation pipeline involves the following steps:

- Each input image is paired with its respective annotation file.
- The annotation file defines the regions of interest using polygons associated with specific class labels.
- These polygons are rasterized to create the segmentation mask corresponding to each image.

During preprocessing, certain cases are handled carefully to ensure data quality:

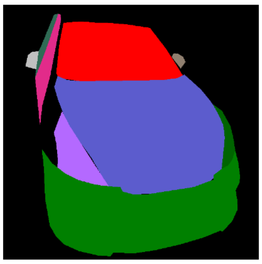
- Annotation files that are empty (i.e., contain no polygon information) are discarded along with their corresponding images.
- In some instances, segmentation masks contain overlapping segments. Such overlaps may introduce ambiguity or noise in the training process.
- To maintain high-quality training data, all such samples with overlaps or invalid annotations are removed from the dataset.

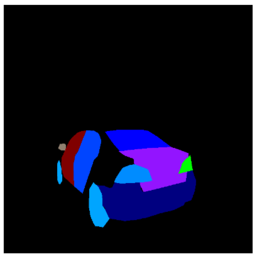
This preprocessing ensures that only valid, well-annotated image-mask pairs are used for training the segmentation model.

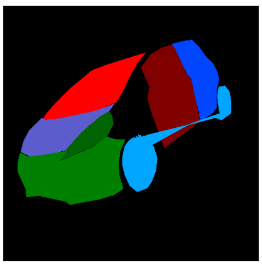
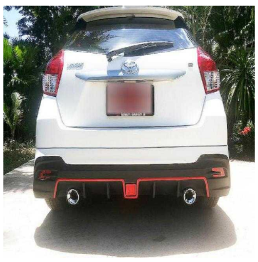
Dataset Summary

- Total Masks Generated: 3833
- Masks Removed Due to Blank or Invalid Annotations: 147
- Masks Removed Due to Overlapping Segments: 173
- **Remaining Valid Masks: 3513**

Example of images and their segmentation masks:




Task 2: Train an Image Segmentation Model

Overview

This architecture is a hybrid of Vision Transformers (ViT) and a UNet-style decoder, tailored for semantic segmentation. It takes an input image and produces a pixel-wise classification map, where each pixel is assigned to a specific class (e.g., door, wheel).

It combines:

- **CNNs** for local feature extraction
- **Transformers** for capturing long-range dependencies
- **Decoder network** for upsampling and spatial reconstruction

1. Encoder

The encoder consists of two main components:

- **CNN Feature Extractor:** A few convolutional layers that extract local features from the input image.
- **Patch Embedding:** The CNN output is split into non-overlapping patches. Each patch is projected into an embedding vector using a Conv2D layer with kernel size and stride equal to the patch size.
- **Positional Encoding:** Positional information is added to the patch embeddings using learnable positional embeddings.
- **Transformer Blocks:** A series of Transformer blocks process these patch embeddings. Each block includes:
  - Layer Normalization
  - Multi-Head Self Attention
  - Feedforward Network
  - Residual connections

  The self-attention mechanism enables the model to understand relationships between distant patches in the image.

2. Decoder

The decoder consists of a series of transpose convolution (deconvolution) layers and convolutional layers:

- The output of the transformer (a compact feature map) is upsampled in multiple stages.
- After each upsampling, the spatial resolution is increased and a convolutional block refines the output.
- This mirrors the typical UNet decoder which gradually reconstructs high-resolution feature maps from the low-resolution encoded features.

3. Output Head

The final layer is a 1×1 convolution that maps the feature map to the desired number of output classes (e.g., 1 for binary segmentation). This gives a per-pixel class prediction.

Why This Architecture?

- **CNNs** efficiently capture local textures and patterns.
- **Transformers** model global context, allowing distant parts of the image to influence each other.
- **UNet-style decoder** reconstructs the spatial structure necessary for accurate segmentation.
- **Patch embeddings with positional encodings** enable the transformer to work effectively on image data while preserving spatial information.

Together, this hybrid architecture benefits from both local and global context understanding, which is ideal for pixel-level tasks like semantic segmentation.

To evaluate the model's ability to generalize, we tested it on a holdout evaluation dataset comprising images that were not seen during training. This ensures that performance metrics reflect the model's capacity to generalize beyond the training data.

Computational Resources

The training was conducted using a single NVIDIA GeForce GTX 1080 Ti GPU. The total training time was approximately 6 hours for 1000 epochs, which demonstrates the feasibility of training the model on a mid-range consumer GPU within a reasonable time frame.

Evaluation Metrics

The performance of the segmentation model was assessed using the following standard metrics:

- **Intersection over Union (IoU)**: Measures the overlap between predicted and ground truth masks.
- **Dice Coefficient**: A similarity measure that balances precision and recall for segmentation tasks.
- **Pixel Accuracy**: The percentage of correctly classified pixels over the total number of pixels.

Evaluation Results

Model performance on the unseen evaluation dataset is summarized as follows:

- **Test IoU**: 0.1487
- **Test Dice Coefficient**: 0.2255
- **Test Pixel Accuracy**: 0.9550

Evaluation Approach

For each image in the evaluation dataset, the model generated a predicted segmentation mask. These masks were compared to the ground truth using the evaluation metrics listed above. The evaluation loop ensured batch-wise processing with appropriate interpolation to match mask dimensions, followed by accumulation and averaging of metric scores. This methodology ensures a reliable and reproducible assessment of the model's quality.