# Task 1: Dataset Preparation using Python

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

Md Farhad Hussain

## 1 Overview

This project processes a COCO dataset to generate segmentation masks suitable for training segmentation models. The masks can be produced as either binary or multi-class based on a command-line flag.

## 2 Dataset Preparation

For this project, we use the COCO 2017 dataset, which provides large-scale object detection, segmentation, and captioning data. The following script was used to download the training and validation images, along with the corresponding annotations:

```
mkdir -p coco_dataset && cd coco_dataset
wget http://images.cocodataset.org/zips/train2017.zip
wget http://images.cocodataset.org/zips/val2017.zip
wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip
unzip train2017.zip && unzip val2017.zip && unzip annotations_trainval2017.zip
```

## 3 Design Decisions

- **Efficient Data Handling:**
  - The script reads COCO annotations from a JSON file and processes only image metadata, avoiding full image loading to ensure scalability with large datasets.
  - COCO API functions (`pycocotools`) are used for accurate and efficient decoding of segmentation data.
- **Mask Generation Strategy:**
  - **Binary Masks:** Generated when the `-binary` flag is set, with regions of interest marked as 1.
  - **Multi-class Masks:** Each category is mapped to a unique integer label using a category-to-label mapping.
  - **Group Masks:** Optional grouping logic resolves overlapping annotations using a custom priority scheme (e.g., prioritizing 'living' entities over 'household' or 'electronics').
- **Handling Overlaps:**
  - When segmentations overlap, the later annotations overwrite earlier ones, unless grouping is enabled.
- **Robust Edge Case Handling:**
  - **Unknown Categories:** Unmapped categories are defaulted to the background class.
  - **Missing or Malformed Segmentations:** Such entries are skipped with a warning logged.
  - **RLE Decoding Errors:** Graceful error handling and logging ensure that decoding failures don't halt execution.

## 4 Implementation

COCOC 2017 has 80 unique classes.

Tabell 1: Unique classes present in COCO 2017 segmentation annotation

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| person | bicycle | car | motorcycle | airplane | bus | train | truck | boat | traffic light | fire hydrant |
| elephant | bear | zebra | giraffe | backpack | umbrella | handbag | tie | suitcase | frisbee | skis |
| wine glass | cup | fork | knife | spoon | bowl | banana | apple | sandwich | orange | broccoli |
| dining table | toilet | tv | laptop | mouse | remote | keyboard | cell phone | microwave | oven | toaster |

80 unique classes is too many for the small dataset 3000-8000. So grouped theses unique classes to 9 relevant group and one background.

Tabell 2: Relevant groups

| Category | Objects |
|---|---|
| **Living Beings** | person, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe |
| **Vehicles** | bicycle, car, motorcycle, airplane, bus, train, truck, boat |
| **Urban Objects** | traffic light, fire hydrant, stop sign, parking meter, bench |
| **Sports** | frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket |
| **Personal Accessories** | teddy bear, hair drier, toothbrush, backpack, umbrella, handbag, tie, suitcase |
| **Kitchen Items** | bottle, wine glass, cup, fork, knife, spoon, bowl, sandwich, broccoli, carrot, hot dog, pizza, donut, cake |
| **Fruits** | banana, apple, orange |
| **Electronics** | tv, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator |
| **Household** | book, clock, vase, scissors, chair, couch, potted plant, bed, dining table, toilet |

| Category | Priority Weight |
|---|---|
| living | 4.0 |
| vehicles | 3.0 |
| urban | 2.5 |
| sports | 1.5 |
| personal | 1.0 |
| kitchen | 2.0 |
| fruit | 2.0 |
| household | 1.0 |
| electronics | 2.5 |

Tabell 3: Priority Weights for Object Categories

The priority weights (e.g., `living`: 4.0, `vehicles`: 3.0) are used to resolve pixel-level conflicts when multiple objects overlap in the same region. During mask generation, if overlapping occurs, the object with the higher priority weight is preserved in the final mask. This approach ensures that semantically important classes (e.g., humans or animals) take precedence over less critical ones (e.g., sports equipment or household items), maintaining the relevance of key objects in the scene.

The script *`generate_masks.py`* is modular and performs the following tasks

- Decoding segmentation annotations.
- Generating and saving masks.
- Saving the list of unique class names.

Logging is utilized for improved traceability and debugging. Batch processing supports datasets in batches.

The following terminal execution will create the mask for training and validation respectively.

```
python3 generate_masks.py --images_dir coco_dataset/train2017 \
  --annotations_file coco_dataset/annotations/instances_train2017.json \
  --output_dir coco_dataset/masks_train --max_images 5000

python3 generate_masks.py --images_dir coco_dataset/val2017 \
  --annotations_file coco_dataset/annotations/instances_val2017.json \
  --output_dir coco_dataset/masks_val --max_images 2000
```

The directory layout of the Task-1.

```
 coco_dataset
    annotations
    classes.txt
    masks_train
    masks_val
```

```
    train2017
    val2017
 generate_masks.py
 README.md

8 directories, 5 files
```

## 5 Results

Figur 1: Image and corresponding segmentation mask.

## 6 Summary

The implemented solution successfully creates segmentation masks from COCO annotations by addressing edge cases such as overlapping annotations and invalid segmentation formats.

# 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 $1 \times 10^{-4}$, 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 ($256 \times 256$).
- 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:

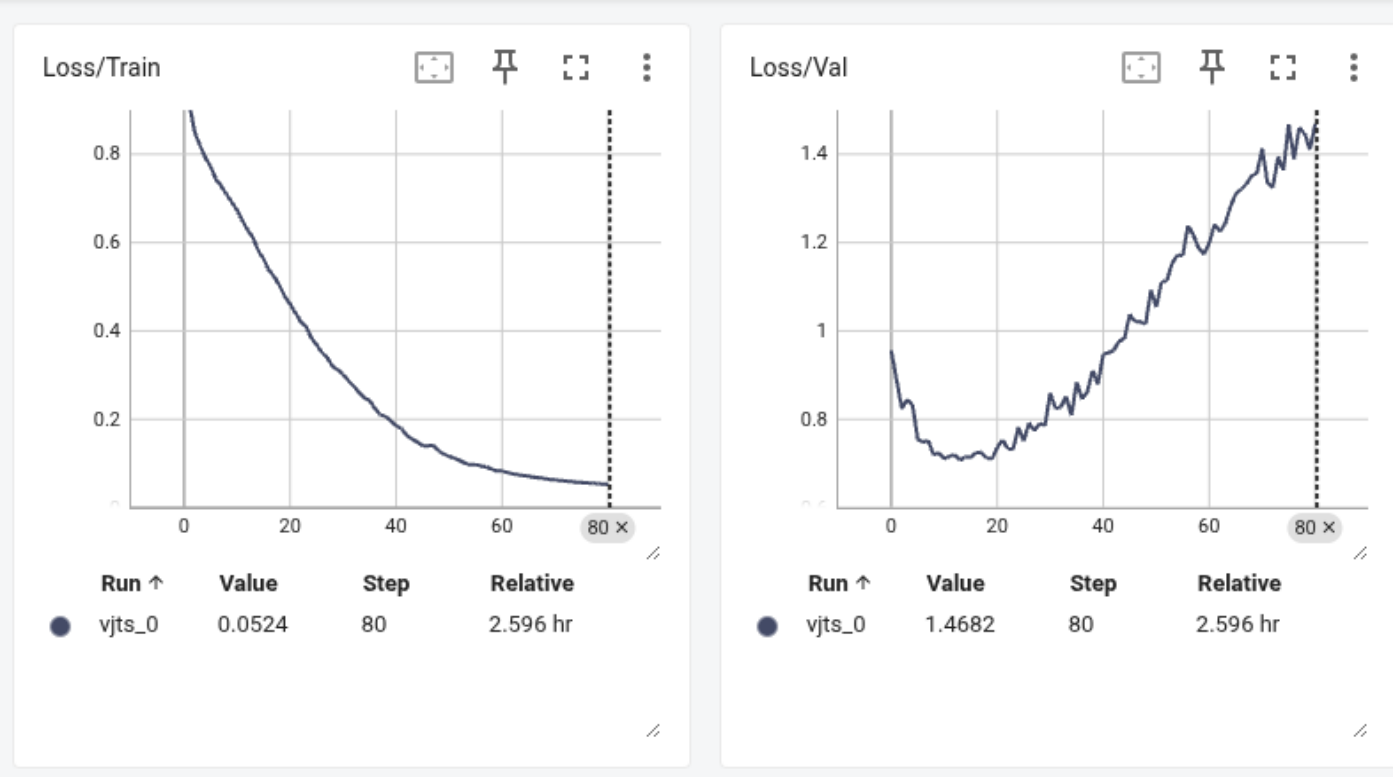
1. Forward propagation to compute segmentation predictions.
2. Loss computation using the chosen loss function.
3. Backward propagation and optimization.
4. Evaluation on a validation set using metrics: Intersection over Union (IoU), Dice coefficient, and pixel accuracy.

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.

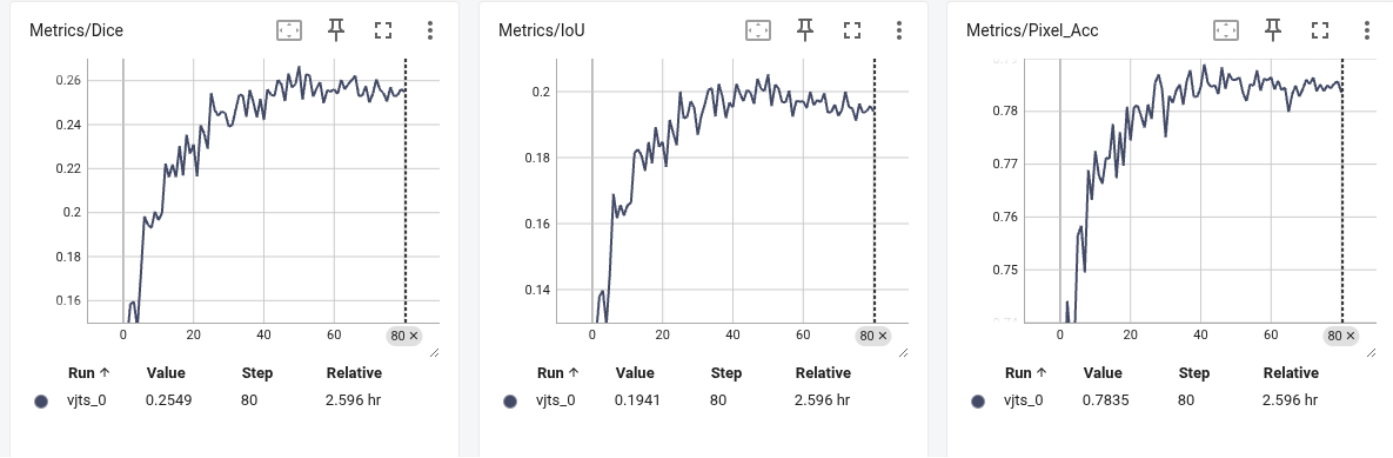
## 5.2 Evaluation Metrics

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).

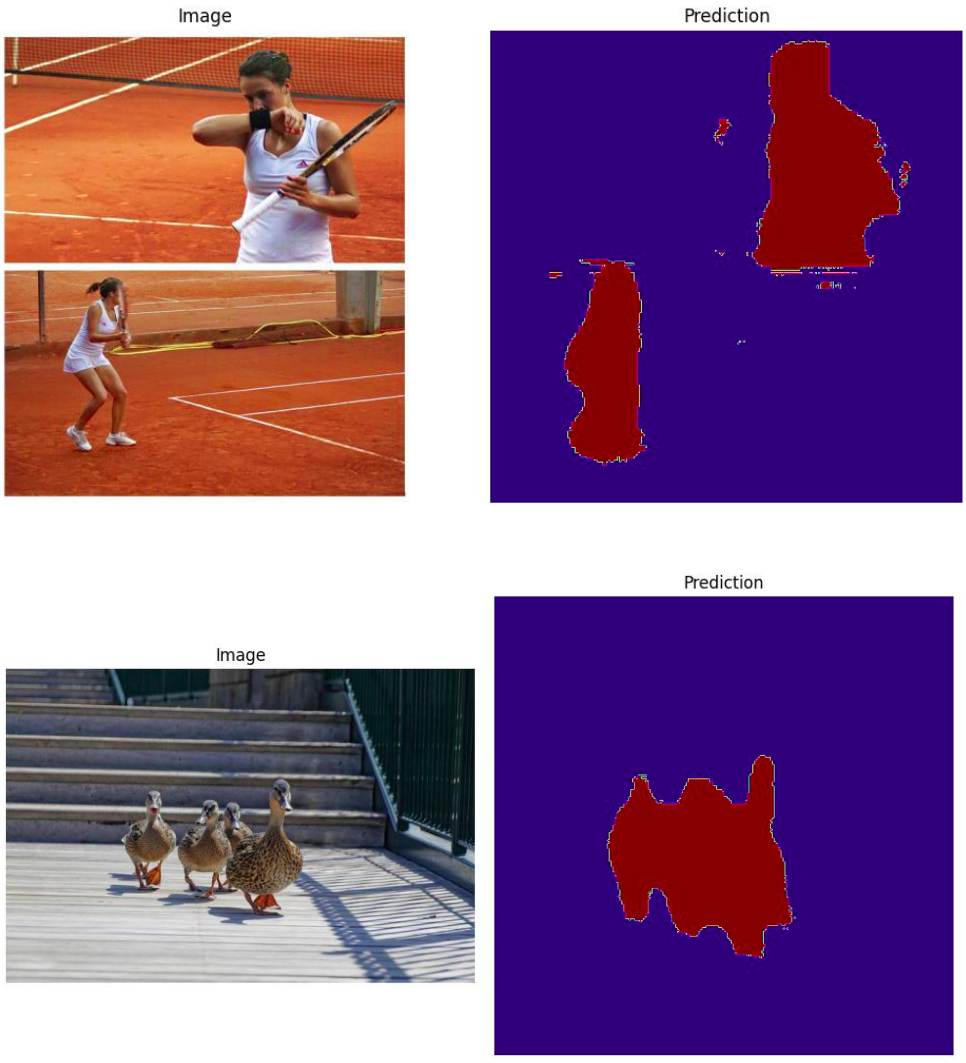
To mitigate overfitting, several approaches can be considered:

- **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.