Salient Object Detection System

Rita Shkodra

Mentor: Dafina Berisha

genpact

Table of Contents

Table of Contents 2
1. Project Overview 3
2. Dataset Description 4
   2.1. Merging and Splitting 4
   2.2. Preprocessing 4
3. Baseline Implementation 5
4. Improved & Final Model 6
   4.1. Data Augmentation 6
   4.2. Learning Rate Scheduler 6
   4.3. Checkpointing 6
   4.4. Updated Setup 6
5. Final Results & Model Comparison 8
6. Challenges & Model Limitations 9
7. Conclusion 10

1. Project Overview

The purpose of this assignment was to design and implement a Salient Object Detection(SOD) system and choosing one of the following datasets to work with: DUTS, ECSSD, MSRA10K, or SALICON. After researching, I decided DUTS was a reasonable choice. The work involved implementing key deep learning components:

- Data preprocessing and loading
- Custom PyTorch Dataset/DataLoader classes
- A full training loop with loss computation and optimizer updates
- Validation and testing procedures
- Segmentation metrics: Precision, Recall, F1-score, IoU
- A baseline UNet model
- An improved version with additional enhancements
- A bonus feature—model checkpointing—was also included.

This report walks through the entire pipeline, explains the reasoning behind key architectural choices, and compares the performance of the baseline and improved approaches.

2. Dataset Description

The experiments in this projects were conducted using the DUTS dataset, which is a widely recognized benchmark for saliency detection research. It is split into two main categories: DUTS-TR for training and DUTS-TE used for model evaluation.

2.1. Merging and Splitting

Seeing as the assignment required an equal split of the data, 70% for training, 15% validation data and 15% testing for model evaluation, the best way to go ahead was merging both TR and TE images, therefore getting 15,572 images in total. These images were then split correctly into:

- 7387 images for training
- 1583 images for validation
- 1583 images for testing

Each sample contains and RGB input image and a corresponding binary saliency mask indicating salient object regions. The images vary in terms of : object scale, background clutter, visual complexity, and illumination.

2.2. Preprocessing

Both images and masks underwent a series of:

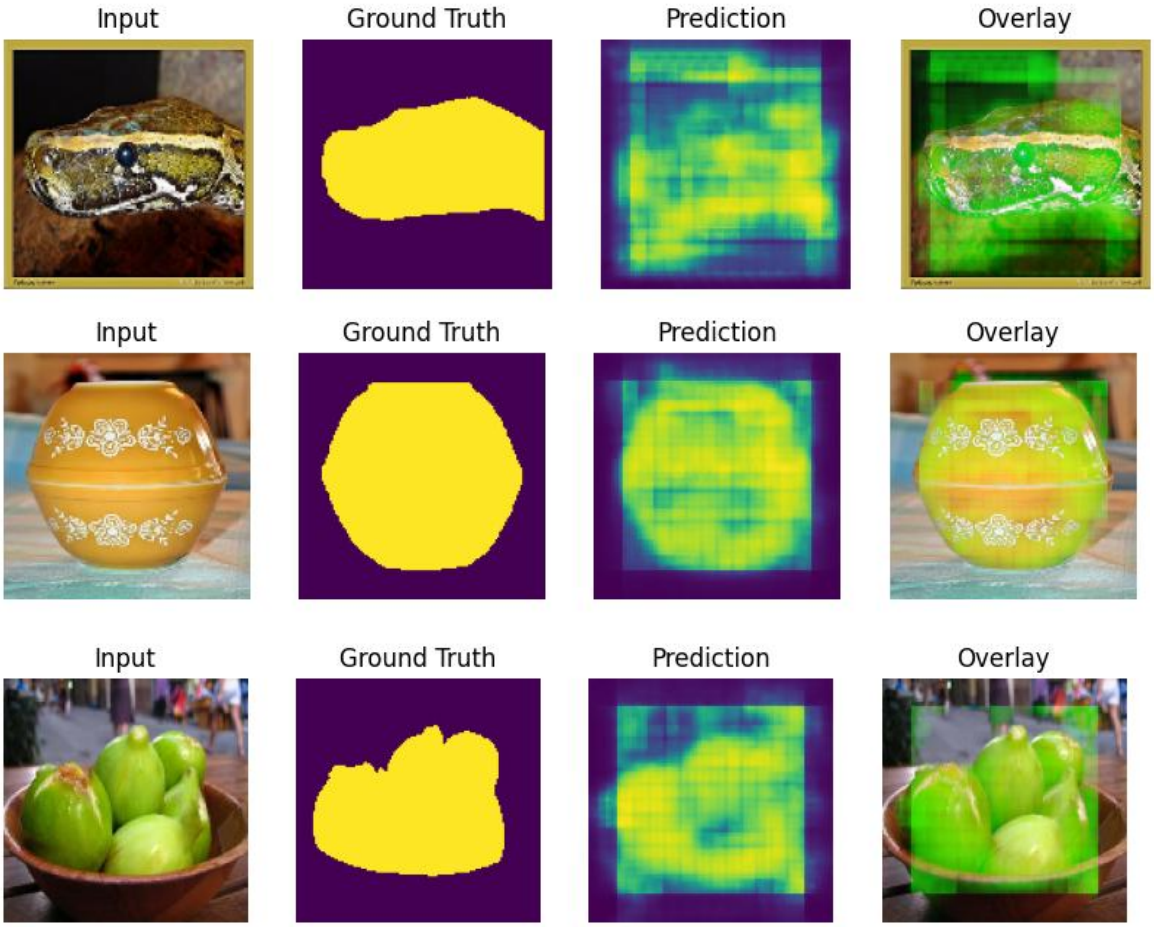
- Resizing to a fixed resolution
- Normalization (pixel values)
- Conversion to PyTorch tensors
- Mask preprocessing (thresholding and float conversion)

3. Baseline Implementation

The baseline system was built using a straightforward and minimal approach to ensure a clean reference before introducing more advanced features. It used a standard UNet model with a simple encoder–decoder structure for binary saliency prediction.

Preprocessing in the baseline was intentionally limited to resizing images and masks, normalizing pixel values, and converting data into PyTorch tensors. No data augmentation or regularization techniques were used, allowing the model to learn directly from the raw dataset distribution.

Training was performed using Binary Cross-Entropy loss and the Adam optimizer with a constant learning rate. The baseline was trained for 20 epochs using small mini-batches, and validation was run after each epoch to monitor performance. No learning rate adjustments, regularization strategies, or checkpointing mechanisms were included. The resulting performance served as the foundation against which all subsequent improvements were measured.



4. Improved & Final Model

Once the baseline pipeline was fully functional, I improved the training setup by adding several enhancements directly into the code. These improvements focused on increasing the model's robustness, improving optimization behaviour, and ensuring that the best-performing version of the model was saved during training.

4.1. Data Augmentation

To increase dataset variability and reduce the model's reliance on specific patterns,a series of augmentations were applied to the training split. These included:

- **Random Horizontal Flip** to increase symmetry variance
- **Random Rotation** to reduce sensitivity to orientation
- **ColorJitter** to improve robustness to lighting and contrast
- **RandomResizedCrop** to expose the model to scale variation

These transformations were implemented using torchvision.transforms, making sure that all augmentations were applied consistently to both the image and its corresponding saliency mask.

4.2. Learning Rate Scheduler

To improve the training dynamics, I added a ReduceLROnPlateau scheduler. Instead of training with one fixed learning rate from start to finish, the scheduler monitored the validation loss and reduced the learning rate if improvements stopped.

4.3. Checkpointing

To meet the bonus requirement, a checkpointing mechanism was introduced. The training loop monitored validation IoU, and when a higher IoU was achieved, the model's weights were saved:

```
if val_iou > best_iou:
    torch.save(model.state_dict(), "best_model.pth")
```

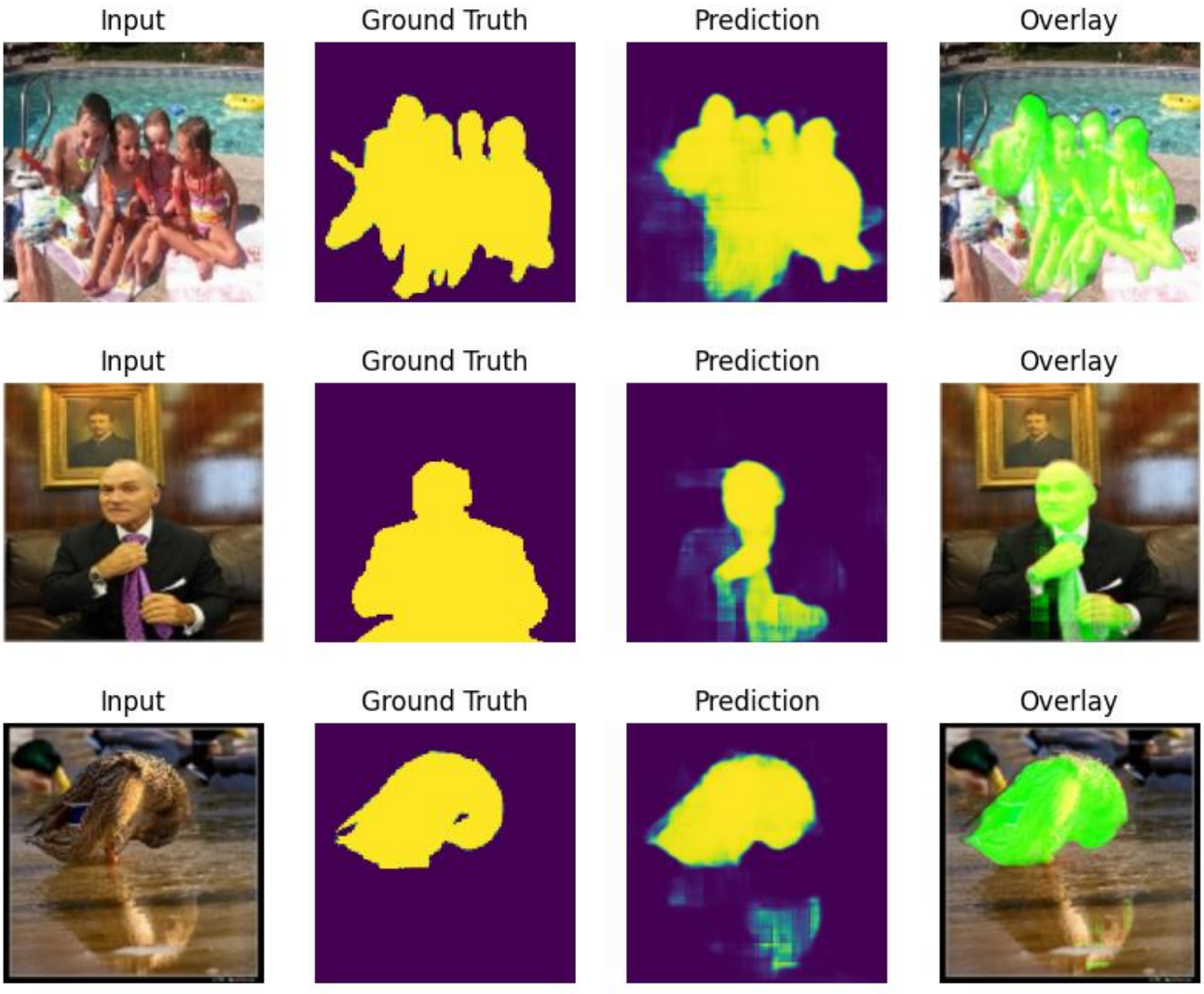
This ensured that the final evaluation used the highest-performing model.

4.4. Updated Setup

The improved training configuration included:

- The same UNet architecture
- BCEWithLogitsLoss for stable optimization
- Adam optimizer
- ReduceLROnPlateau scheduler
- A 25-epoch training schedule
- Batch size of 4
- Dataset split: 7,387 / 1,583 / 1,583

These improvements significantly strengthened the model compared to the baseline.



5. Final Results & Model Comparison

Table 1: Model Comparison

Metric	Baseline	Improved	Improvement
Precision	0.6480	*0.8471*	+0.1991
Recall	0.7639	0.8232	+0.0593
F1-Score	0.6533	0.8110	+0.1577
IoU	0.5162	0.7227	+0.2065

The difference between the baseline and improved model is clearly reflected in the final metrics. The improved model achieves much higher Precision (**0.8471**) compared to the baseline (**0.6480**), showing that it made far fewer false-positive predictions. Recall also increased to **0.8232**, meaning the model captured a larger portion of the true salient regions. These results confirm that the improvements we introduced into the training process were effective. Data augmentation helped the model generalize better, the scheduler improved long-term optimization, and the checkpointing system ensured that the best version of the model was saved.

6. Challenges & Model Limitations

During the evaluation of the baseline model, several consistent issues became clear. The most noticeable problem was the high number of false positives, especially in backgrounds with strong textures or bright regions. This directly explains the baseline Precision score, which stayed low. When reviewing the predicted masks, the model often highlighted areas that were not actually salient, and the boundaries of the detected objects were usually not precise.

The baseline also struggled with smaller or thinner objects. In many cases, these regions were only partially segmented or missed entirely. This contributed to the lower IoU score, since the predicted masks did not match the ground-truth shapes accurately. Overall, the baseline lacked generalization and tended to overfit to the specific patterns in the training set.

After adding the improvements such as augmentations, the ReduceLROnPlateau scheduler, and checkpointing — many of these issues decreased noticeably. The improved model produced much cleaner masks, fewer false detections, and significantly sharper edges. This improvement was reflected in the IoU jump from 0.5162 to 0.7227.

Even with these improvements, there are still some limitations. The model sometimes misses very small salient regions, and in images with low contrast, the predictions can still be slightly worse. These cases show where additional changes or loss functions might help in future versions, but overall, the error patterns are far less frequent than in the baseline.

7. Conclusion

This project successfully delivered a complete Salient Object Detection pipeline built in PyTorch, starting from a simple baseline and improving it step by step. The final model performs significantly better than the initial version across all metrics, especially in Precision, F1-score, and IoU. These improvements clearly show the impact of the changes we introduced, including data augmentation, the learning rate scheduler, and the checkpointing mechanism.

All requirements of the assignment were met, including the bonus feature. The project covers the full pipeline end-to-end, from dataset preparation to evaluation on DUTS-TE. It also demonstrates a clear understanding of how training dynamics, data preprocessing, and model-saving strategies affect final performance.

Overall, the improved system forms a solid foundation for future work. Adding more advanced architectures or hybrid loss functions could strengthen performance even further, but the current implementation already represents a strong and reliable SOD mode