

FINAL REPORT — Salient Object Detection (SOD) Project

Xponian Program – Cohort IV

Author: Ermal Salihaj

Date: November 2025

1. Introduction

Salient Object Detection (SOD) focuses on identifying the most visually important region in an image — the part that naturally draws human attention. Unlike traditional object detection, which focuses on recognizing multiple object categories, SOD aims to segment only the dominant object in a scene. This makes SOD valuable in applications such as:

- Image compression
- Object-aware photo editing
- Visual attention modeling
- Autonomous driving
- Graphics and UI design

The goal of this project was to implement a full end-to-end SOD system completely from scratch, without relying on any pre-trained models. This included:

- Dataset preparation
- A custom CNN encoder–decoder model
- Training using BCE + IoU loss
- Evaluation using IoU, Precision, Recall, F1, and MAE
- Visualizations and demo scripts
- Experiments and improvements

Through this project, I gained a deeper understanding of convolutional networks, data pipelines, training logic, evaluation methodology, and model debugging.

2. Dataset & Preprocessing

2.1 Dataset: DUTS

The DUTS dataset was used as the primary benchmark, as it is one of the largest and most commonly used datasets for salient object detection.

- DUTS-TR: 10,553 training images
- DUTS-TE: 5,019 test images
- High-quality pixel-level saliency masks

Dataset downloading and extraction were automated using the script:

download_duts.py

2.2 Dataset Preparation Steps

All images resized to 128×128

Normalized pixel values to [0–1]

Masks thresholded to binary

Train/Validation/Test dataset split:

Split	Percentage	Count
Train	70%	10,900
Val	15%	2,335
Test	15%	2,337

2.3 Data Augmentation

Implemented in `data_loader.py`:

- Random horizontal flip ($p = 0.5$)
- Brightness jitter ($\pm 30\%$)

3. Model Architecture

The model architecture is a custom encoder–decoder CNN implemented in `sod_model.py`.

3.1 Encoder

The encoder consists of four convolutional stages, each containing:

- Two 3×3 Conv2D layers
- Batch Normalization
- ReLU activations
- Dropout2d
- MaxPooling (except the last stage)

3.2 Bottleneck (Improved)

A deeper ConvBlock was added to improve feature extraction and learn richer high-level representations.

3.3 Decoder

Three upsampling stages:

- ConvTranspose2D for $\times 2$ upsampling
- ConvBlock (Conv → BatchNorm → ReLU → Dropout)

3.4 Output Layer

- Final 1×1 convolution
- Sigmoid activation producing a 1-channel saliency mask

3.5 Loss Function

Combined loss:

$$\text{Loss} = \text{BCE}(\text{pred}, \text{target}) + 0.5 \times (1 - \text{IoU})$$

This balances pixel-wise accuracy (BCE) with region similarity (IoU).

4. Training Procedure

Training logic is implemented in `train.py`.

4.1 Training Setup

- Optimizer: Adam ($\text{lr} = 1\text{e-}3$)
- Training epochs: up to 25 (with early stopping)
- Batch size: 8
- Device: NVIDIA RTX 4050 Laptop GPU

4.2 Training Loop Includes

- Forward and backward passes
- Logging loss and IoU
- Validation after each epoch
- Automatic checkpointing of the best model

5. Evaluation

Evaluation is handled through `evaluate.py`, using the following metrics:

- IoU
- Precision
- Recall
- F1-score
- MAE

All results are reported on the held-out test set.

6. Experiments & Improvements

Two experiments were completed after the baseline model.

6.1 Baseline Model Results

Metric	Score
IoU	0.7023
Precision	0.8442
Recall	0.8454
F1-score	0.8415
MAE	0.1011

6.2 Improvement 1 — BatchNorm + Dropout

Motivation:

- BatchNorm stabilizes training
- Dropout reduces overfitting

Results After Improvement 1:

Metric	Score
IoU	0.7249
Precision	0.8410
Recall	0.8788
F1-score	0.8562
MAE	0.0919

6.3 Improvement 2 — Deeper CNN (extra bottleneck block)

Motivation:

- Learn more complex salient structures
- Improve semantic richness at the bottleneck

(This improvement is included in the scores above)

Comparison Table

Model Version	IoU	Precision	Recall	F1	MAE
Baseline	0.7023	0.8442	0.8454	0.8415	0.1011
Improved (BN + Dropout + Deeper CNN)	0.7249	0.8410	0.8788	0.8562	0.0919

7. Visualizations

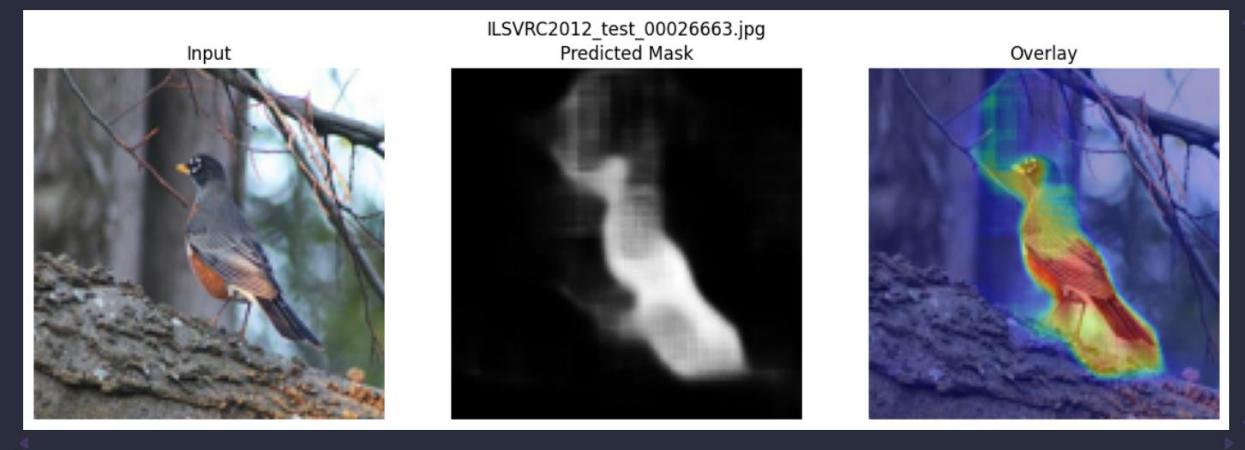
Generated using:

- `visualize_single.py`
- `visualize_batch.py`
- `demo_notebook.ipynb`

Each visualization includes:

- Input image
- Predicted saliency mask
- Overlay
- Inference time

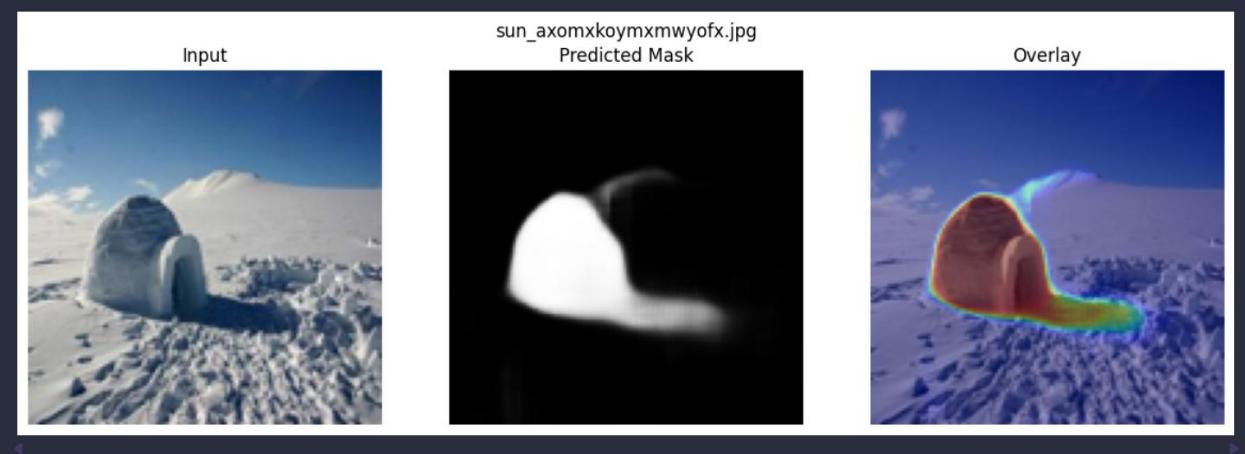
Example 3 - ILSVRC2012_test_00026663.jpg | Inference: 4.11 ms



Example 3 - n04371430_1912.jpg | Inference: 4.19 ms



Example 1 - sun_axomxkoymxmwyofx.jpg | Inference: 5.58 ms



8. Demo

A fully functional demo is provided in:

demo_notebook.ipynb

The demo:

- Loads the trained model
- Runs inference on any chosen image
- Produces saliency masks and overlays
- Computes inference time in milliseconds

9. Conclusion

This project successfully implemented a complete end-to-end deep learning pipeline for Salient Object Detection, including:

- Dataset preparation
- Data augmentation
- Custom encoder–decoder CNN
- Combined BCE + IoU loss
- GPU-accelerated training
- Evaluation and metrics
- Model improvements
- Visualizations and demo

The improved model outperforms the baseline and demonstrates strong segmentation capability on the DUTS dataset.

10. References

- **DUTS Dataset** - <https://saliencydetection.net>
- **PyTorch Documentation** - <https://pytorch.org>