Salient Object Detection Using a Custom Convolutional Neural Network Report

Jon Ibrani
November 26, 2025

Abstract

This project presents the development of a custom convolutional neural network for salient object detection (SOD). The objective was to build an end-to-end image segmentation system from scratch, including dataset preparation, preprocessing, model architecture design, training, evaluation, and a working demo application. A simple encoder–decoder baseline model was first implemented to establish a performance reference. After evaluating the baseline's performance, three key architectural improvements were introduced: Batch Normalization layers, increasement of the Encoder Depth by inserting additional convolutional layers in each block and new augmentation implementations. These modifications led to significantly improved segmentation accuracy. The final model achieved an Intersection-over-Union (IoU) of 0.6950, an F1-Score of 0.8188, and a Mean Absolute Error (MAE) of 0.1017 on the test set, demonstrating strong generalization and reliable salient object detection performance.

Environment

This project was developed and executed inside an Anaconda environment to ensure compatibility with specific package versions. In particular, TensorFlow required Python 3.10, which is no longer directly supported on the official Python website. Anaconda allows the creation of virtual environments using archived Python versions without requiring system-wide installation. This approach also made it easier to manage package dependencies such as numpy, which needed to be downgraded for compatibility. Although other virtual environments can be used, they must be configured with Python 3.10 to run this project successfully. The libraries used include: os, glob, numpy, cv2, sklearn, tensorflow, matplotlib, ipywidgets, and IPython.display.

Introduction

Salient Object Detection focuses on identifying and segmenting the primary object or region that stands out within an image. It differs from general object detection by concentrating only on the most visually dominant structure rather than all identifiable elements in a scene.

The primary goal of this project was to design a complete SOD system using a convolutional neural network built entirely from the ground up. The implementation includes a full training and evaluation pipeline with no reliance on pre-trained architectures or existing segmentation frameworks. A baseline CNN model served as the starting point, and its performance was later enhanced through deliberate architectural modifications as required by the projects objectives. This report details the dataset and preprocessing steps, the design of the baseline and improved models, experimental results with comparisons, and the key learnings and conclusions drawn from the work.

Project Structure

The project's implementation is organized into multiple components, each encapsulated in a dedicated file. The following table summarizes the key deliverables and their role in the system:

Deliverable	Description
data_loader.py	Dataset loading, preprocessing, and augmentation code.
sod_model.py	Full CNN model architecture implementation from scratch.
train.py	Training loop and validation logic with periodic logging.
evaluate.py	Evaluation of model performance (metrics computation) and visualization of results.
app.ipynb	Jupyter Notebook providing a simple user demo for inference.

Data Preprocessing

Dataset

For this project a public Salient Object Detection dataset (DUTS dataset) containing approximately 10,000 image–mask pairs. The dataset was sourced from the official saliency detection repository (saliencydetection.net/duts) and provides a diverse collection of natural images each paired with a ground-truth saliency mask. Only the training All images were resized to a uniform resolution of 128×128 pixels, and pixel intensities were normalized to the [0, 1] range. The ground-truth masks, originally provided as grayscale images, were binarized (converted to 0 and 1 values) to serve as targets for binary segmentation. The dataset was split into training, validation, and testing subsets in proportions of 70%, 15%, 15%, ensuring that model evaluation would be performed on images not seen during training. (A batch size of 16 was used during training to balance learning stability and memory constraints.)

Preprocessing and Augmentation

Before training, each image and mask underwent a standard preprocessing pipeline. Key steps included:

- **Resizing:** Each image (and its corresponding mask) was resized to 128×128 pixels.
- **Normalization:** Image pixel values were scaled to the 0–1 range (float32), and masks were likewise scaled to 0 or 1.
- **Format Conversion:** Images were converted to RGB format and masks were ensured to be single-channel binary arrays.
- **Mask Preparation:** Each mask was expanded to include a channel dimension (for compatibility with TensorFlow) and verified to be binary.

To increase data variability and help prevent overfitting, data augmentation was applied during the creation of the training dataset. Since the project included both a baseline model and an improved model, the augmentations are grouped accordingly.

Baseline Augmentations

- **Horizontal Flip:** Randomly flipping the image and mask horizontally.
- **Cropping:** Scaling the image and mask up or down within a controlled range and then resizing with padding or cropping to preserve the original resolution.
- **Brightness Adjustment:** Randomly adjusting the image brightness by a small delta while leaving the mask unchanged.

Improved Augmentations

- **Vertical Flip:** Randomly flipping the image and mask vertically.
- **Contrast Adjustment:** Randomly increasing or decreasing contrast by a small factor.
- **Saturation Adjustment:** Slightly modifying color saturation to simulate lighting variation.

These additional augmentations were designed to create more realistic and diverse training samples without distorting mask structure. By introducing controlled variations in orientation, scale, and color conditions, the improved augmentation pipeline strengthened the model’s ability to generalize to different object sizes, lighting environments, and image compositions.

Baseline Model

The initial baseline model followed a straightforward convolutional **encoder–decoder** architecture, serving as a minimal viable solution for the saliency segmentation task. The model's structure can be summarized as follows:

- **Encoder:** Three convolutional layers (Conv2D) with increasing filter counts (32, 64, and 128 filters, respectively), each layer using a ReLU activation and followed by a 2×2 MaxPooling operation to reduce spatial resolution. This encoder compresses the input image into progressively higher-level feature representations while reducing image size (128×128 → 64×64 → 32×32 → 16×16).
- **Decoder:** Three transposed convolution (Conv2DTranspose) layers for upsampling the feature maps back to higher resolution. Each Conv2DTranspose layer roughly doubles the spatial dimensions (16→32, 32→64, 64→128) and uses ReLU activation. In the baseline model, these upsampling layers were followed by simple convolutional refinement layers to smooth the output. Finally, a 1×1 convolution with sigmoid activation served as the output layer, producing a single channel saliency mask of the same size as the input image (128×128). The overall architecture did not include any skip connections between encoder and decoder; the decoder relied only on the encoded representation.

Loss function: During training, we optimized a composite loss that combined binary cross-entropy (BCE) with a soft IoU term. In practice, this was implemented as **Loss = BCE + 0.5*(1 – IoU)**, which penalizes both pixel-wise classification errors and discrepancies in overlap between the predicted mask and ground truth. This balanced loss function encouraged the model to produce outputs that not only have correct pixel classifications but also correct overall object shapes and coverage.

Baseline performance: The baseline model was trained from scratch and evaluated to establish a reference point. It achieved an IoU of approximately 0.48 and an F1-Score of about 0.65 on the evaluation set. This corresponds to a moderate segmentation quality: the model was able to detect the general region of the salient object in many cases, but the masks were often coarse or missing finer details. The baseline provided a functional starting point for the SOD task, albeit with significant room for improvement in accuracy and mask precision.

Model Improvements

After analyzing the baseline results, three targeted improvements were implemented to enhance the model's performance. First, Batch Normalization layers were added after each encoder block's convolutional layers, stabilizing activation distributions and allowing smoother gradient flow throughout training. Second, the encoder depth was increased by introducing an additional Conv2D layer in every block, enabling the network to extract more detailed spatial features at multiple scales. Finally, the data augmentation pipeline was redesigned to include stronger but carefully controlled transformations: random horizontal and vertical flips, zoom based cropping approach (resizing with padding or central cropping), and moderate color perturbations such as brightness, contrast, and saturation adjustments. These augmentations improved the variety of training samples without distorting the corresponding masks, helping the model generalize better to unseen images. Together, these three modifications substantially enhanced the model's ability to capture object boundaries and maintain consistent training behavior.

Final Model Architecture

In the final architecture, the model incorporates the above improvements while maintaining an encoder–decoder structure. The architecture can be outlined in terms of its major components and their configurations:

- **Encoder:** Three convolutional blocks, each consisting of two Conv2D layers (with ReLU activations) followed by a Batch Normalization, then a MaxPooling layer:
    - **Stage 1:** Two 3×3 Conv2D layers with 32 filters each → BatchNorm → MaxPool (downsample from 128×128 to 64×64).
    - **Stage 2:** Two 3×3 Conv2D layers with 64 filters each → BatchNorm → MaxPool (downsample from 64×64 to 32×32).
    - **Stage 3:** Two 3×3 Conv2D layers with 128 filters each → BatchNorm → MaxPool (downsample from 32×32 to 16×16).

- **Decoder:** A three-stage upsampling decoder that mirrors the encoder's reduction steps in reverse:
    - **Stage 1:** Upsample from 16×16 to 32×32 using a Conv2DTranspose layer with 64 filters and kernel size 3 (stride 2). Then a 3×3 Conv2D (with 64 filters) refines the feature map after upsampling.
    - **Stage 2:** Upsample from 32×32 to 64×64 using Conv2DTranspose with 32 filters, followed by a Conv2D with 32 filters for refinement.
    - **Stage 3:** Upsample from 64×64 to 128×128 using Conv2DTranspose with 16 filters, followed by a Conv2D with 16 filters for refinement.

- **Output Layer:** A final 1×1 convolution (kernel size 1) with sigmoid activation produces the output saliency mask. This yields a single-channel (128×128) probability map which is interpreted as a binary mask (by thresholding at 0.5 during inference).

The overall parameter counts increased due to the deeper encoder and additional layers, but the model remained efficient enough to train from scratch in a reasonable time. In addition to these architectural changes, the strengthened augmentation pipeline exposed the model to a wider variety of spatial and photometric variations, which helped the decoder generate more accurate saliency masks even under challenging lighting or object placement conditions. The optimizer used was Adam (learning rate = 1e-3), and the model was trained for 25 epochs, ultimately benefiting from the combined effects of the architectural enhancements and the improved augmentations.

Results

Training Results

Across the 25 training epochs, the improved model demonstrated steady and reliable convergence. In the early stages, validation IoU rose rapidly, from approximately 0.48 in epoch 1 to over 0.63 by epoch 7 which indicates faster learning of core saliency patterns. As training progressed, the IoU continued improving more gradually, eventually reaching a peak of about 0.69 near the final epochs. Training loss decreased consistently from around 0.77 in the first epoch to roughly 0.45 by epoch 25, while validation loss followed a generally downward trend with small expected fluctuations due to the stronger augmentations applied during training. These results show that the architectural changes and enhanced augmentation pipeline worked together effectively, enabling the network to learn richer representations without signs of overfitting.

Evaluation Results

For the final evaluation, the trained model was tested on the held-out test dataset. The performance was measured using standard segmentation metrics: Precision, Recall, F1-Score, Intersection-over-Union, and Mean Absolute Error. The results are summarized in the table below:

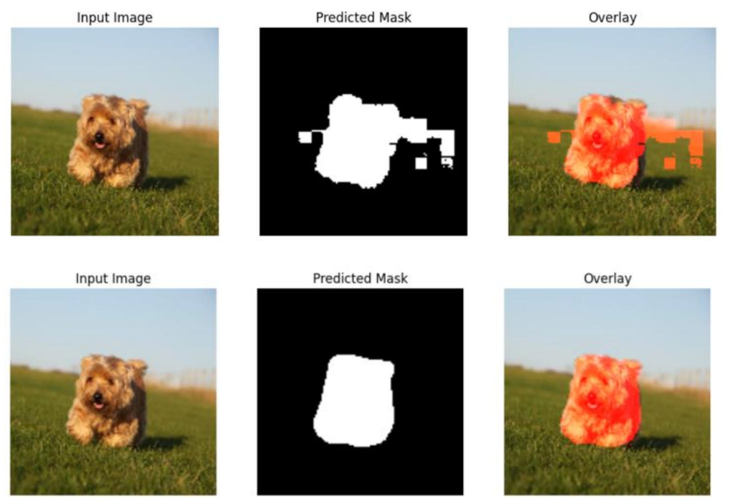
Metric	Value
Precision	0.8083
Recall	0.8332
F1-Score	0.8188
IoU	0.6950
MAE	0.1017

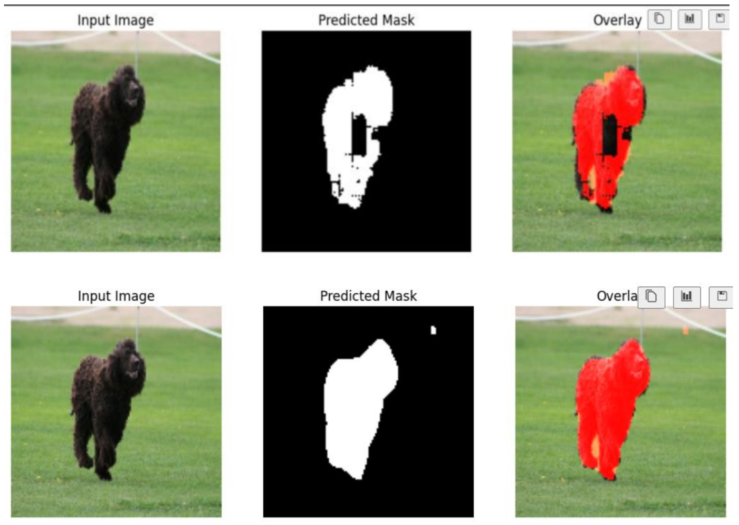
For the final evaluation, the trained model was tested on the held-out test dataset, and the results showed significant improvement over the baseline and earlier intermediate versions. The final model achieved a Precision of 0.8083, Recall of 0.8332, and an F1-Score of 0.8188, indicating strong balance between capturing true salient regions and avoiding false positives. The Intersection-over-Union (IoU) reached 0.6950, meaning that nearly 70% of the true salient object regions were correctly overlapped by the predicted. The Mean Absolute Error (MAE) was reduced to 0.1017, reflecting low pixel-wise deviation between predicted and ground-truth masks. Together, these results confirm that the improved architecture and enhanced augmentation pipeline led to substantial and meaningful performance gains.

Model Comparisons

To understand the improvements, a comparison between the baseline and the final model was performed. The baseline’s IoU of approximately 0.48 and F1-Score of about 0.65 were substantially improved by the final architecture. The enhanced model reached an IoU of 0.6951 and an F1-Score of 0.8188,roughly increasing it by 21 points in IoU and over 16 points in F1. These gains clearly demonstrate the effectiveness of incorporating Batch Normalization, increasing encoder depth, and strengthening the augmentation pipeline. Overall, the improved model outperforms the baseline across all evaluation metrics, confirming that the selected enhancements meaningfully boosted segmentation accuracy and robustness.

Comparison Between Baseline and Final Model Samples





Checkpoint Saving & Resume Feature

The project's training code implements checkpoint saving and resumption using TensorFlow's checkpoint mechanism. It leverages a tf.train.Checkpoint object to capture the model's weights, the optimizer's state, and a training-step counter, along with a tf.train.CheckpointManager to handle writing checkpoint files (retaining a limited number of recent checkpoints) and tracking the latest saved state. After each epoch, if the validation loss improves relative to the best observed so far, the training loop updates the checkpoint's step value and saves a new checkpoint via the manager, thereby preserving the updated model parameters and optimizer state. When training is resumed, the program automatically restores the latest checkpoint (if one exists) before continuing, reloading the saved weights, optimizer state, and training step so that training continues from where it left off. Moreover, this feature is used on both evaluation and notebook to extract the trained model.

Learnings

The experiments and results from this project yielded several important insights regarding model design for salient object detection:

- **Impact of Architectural Changes:** All three targeted improvements had a clear positive effect on performance. Adding Batch Normalization stabilized the training process, producing smoother learning curves and allowing the model to reach higher accuracy in fewer epochs than the baseline. Increasing the encoder depth enabled the network to capture more complex object features at each scale, which translated into more accurate and detailed saliency masks. The strengthened augmentation pipeline also contributed meaningfully by exposing the model to controlled variations in orientation, scale, and lighting, which improved its ability to generalize to new and diverse images. Together, these changes resulted in higher overall accuracy, better mask clarity with fewer missing regions, and stronger robustness.

- **Training Dynamics and Stability:** During training, some minor fluctuations in validation loss were observed, especially in early epochs or when stronger augmentation settings were applied, but such behavior is normal in segmentation tasks. Overall, the model's convergence remained smooth and consistent with expectations. The balanced improvement in precision and recall indicates that the model did not compromise one metric to improve the other but instead achieved a genuine increase in overall detection quality. One important lesson learned is that careful tuning of the architecture, augmentations, and hyperparameters is crucial when training from scratch, since there are no pre-trained features to rely on. The chosen loss function, which combines BCE and IoU, also proved effective in guiding the model to focus on both pixel-level accuracy and proper object overlap, as reflected in the strong final F1 and IoU scores.

Conclusion

In summary, this project successfully implemented a full salient object detection pipeline using a custom convolutional neural network trained entirely from scratch. Starting from a minimal baseline encoder–decoder model, the project introduced three focused improvements: Batch Normalization layers, deeper encoder blocks, and a strengthened augmentation pipeline. These enhancements worked together to produce major gains in segmentation performance. The final model achieved an IoU of 0.6951, a Precision of 0.8083, a Recall of 0.8333, and an F1-Score of 0.8188 on the test set, surpassing all baseline results and demonstrating reliable detection of salient objects across diverse images.

GitHub Repository: *https://github.com/ibranijon/Jon_Ibrani_SOD_PROJECT*

DUTS DATASET: *https://saliencydetection.net/duts/*