

Image Super-Resolution App

Image Super-resolution App – Full
Technical Documentation

By- Arkaprava Roy
14.01.2026

Project Overview

The Image Super-Resolution App is a deep learning–powered web application that enhances low-resolution images using EDSR (Enhanced Deep Super-Resolution Network). The application is built with Streamlit and is designed for both local execution and cloud deployment (Streamlit Community Cloud).

This project demonstrates:

- Practical deployment of deep learning models
- Handling real-world deployment constraints (CPU, memory limits)
- End-to-end ML application lifecycle

Motivation

Low-resolution images are common in:

1. Old photographs
2. Compressed images from the web
3. Surveillance and medical imaging

Traditional interpolation methods (bicubic, bilinear) fail to recover fine details. Deep learning–based super-resolution models like EDSR learn complex mappings to reconstruct high-frequency details, producing visually superior results.

This project aims to:

- a. Provide an interactive, user-friendly interface for super-resolution
- b. Demonstrate practical ML deployment, not just model training

Features

1. Upload PNG / JPG / JPEG images
2. Choose upscale factor (2 \times or 4 \times)
3. Real-time super-resolution inference
4. Before/After comparison slider
5. Download super-resolved image

Tech Stack

1. Frontend

Streamlit – UI framework

streamlit-image-comparison – Before/After slider

2. Backend / ML

PyTorch – Deep learning framework

torchsr – Super-resolution model zoo

EDSR – Pretrained super-resolution model

3. Image Processing

Pillow (PIL) – Image I/O

torchvision.transforms – Tensor conversion

4. Deployment

Localhost – Development & testing

Streamlit Community Cloud – Demo

Project Structure

```
project_root/
  |
  └── temp                         #stores temp images
  └── app.py                        # Main Streamlit application
  └── requirements.txt              # Python dependencies
  └── README.md                     # Basic project overview (optional)
```

Requirements & Dependencies

- Torch
- Torchvision
- Torchsr
- streamlitstreamlit-image-comparison
- Pillow

Python version: python 3.9.x – 3.10.x recommended, newer version creates extreme incompatibilities

Model Architecture (EDSR)

EDSR (Enhanced Deep Super-Resolution Network) is a CNN-based architecture optimized for super-resolution tasks.

Characteristics:

Removes batch normalization layers (improves performance)

Uses residual blocks

Optimized for perceptual quality

Pretrained Models Used:

EDSR ×2

EDSR ×4

EDSR ×8 (Can be used, but removed due to high resource demand as it is deployed on a free cloud server)

Models are loaded using:

```
from torchsr.models import edsr
```

Application Workflow

1: Model Initialization

Models are loaded once using `@st.cache_resource`

Automatically selects CPU or GPU

2: Image Upload

User uploads an image

Image is converted to RGB

3: Upscale Factor Selection

User selects 2× or 4×

4: Inference

Image converted to tensor

Passed through selected EDSR model

Output tensor converted back to image

5: Visualization

Original image displayed

Before/After slider rendered

6: Download

Super-resolved image downloadable as PNG

Cloud Deployment Constraints

Problem:

Streamlit Community Cloud:

- CPU-only
- Limited RAM (~1 GB)
- No GPU acceleration

Observation:

- 2× super-resolution works reliably
- 4× super-resolution may crash for large images due to high memory usage

Possible solution(yet to be implemented):

1. Switch to a better server
2. Use a warning:

```
MAX_PIXELS_4X = 512 * 512
if scale == 4:
    w, h = img.size
    if w * h > MAX_PIXELS_4X:
        st.warning("Image resized for 4x stability on cloud")
    img = img.resize((512, 512))
```

Performance Considerations

Problems:

- a. Memory usage grows quadratically with upscale factor
- b. 4× output contains 16× more pixels than input
- c. Inference time increases significantly on CPU

Optimizations applied:

1. Single-thread execution
2. Model caching
3. Explicit memory cleanup

Deployment

Local Deployment @localhost

Run Locally at project Folder:

```
pip install -r requirements.txt  
streamlit run app.py
```

Server Deployment:

1. Push project to GitHub repository
2. Go to <https://share.streamlit.io>
3. Connect GitHub account
4. Select repository
5. Set app.py as entry point
6. Deploy

Link:

<https://image-super-resolution-app-ibmbu2m5hlc2g2szjtafd.streamlit.app/>

Image Super-Resolution App

Upscale images using deep learning (EDSR)

Upload an image

Drag and drop file here
Limit 200MB per file - PNG, JPG, JPEG

Browse files

Original Image



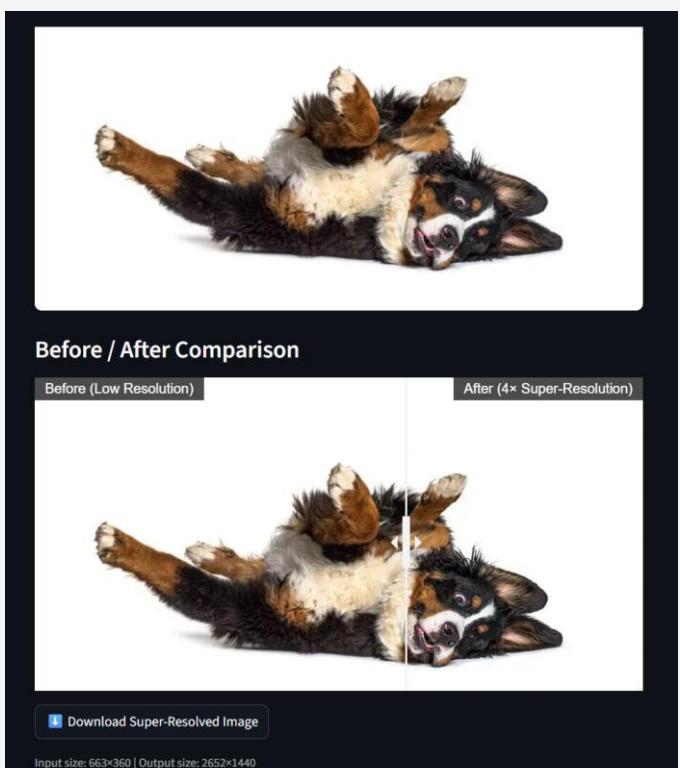
Before / After Comparison

Before (Low Resolution) After (2x Super-Resolution)



Download Super-Resolved Image

Input size: 525x350 | Output size: 1050x700



Future Improvements

Tile-based inference for large images
GPU deployment (Hugging Face / AWS)
Image format optimization
Model selection UI
Better Looking UI
Integration with API for better future scalability

○ Support for ESRGAN / Real-ESRGAN:

The trial for ESRGAN / Real-ESRGAN has already been initiated but soon it is found that, while implementing we experienced same error every time :

ModuleNotFoundError: No module named 'torchvision.transforms.functional_tensor'

Its is found that :

"functional_tensor submodule was deprecated and removed in newer versions of Torchvision, starting from version 0.15.

This module was previously used for tensor-specific image transformations but has been replaced by the functional module."

Same Issue like mine can be found at:

<https://github.com/pytorch/vision/pull/7269>

<https://discuss.pytorch.org/t/no-module-named-torchvision-transforms-functional-tensor/206390>

Deprecate functional_pil and functional_tensor and make them private #7269

Merged NicolasHug merged 2 commits into pytorch:main from NicolasHug:dep_fffff on Feb 16, 2023

Conversation 9 Commits 2 Checks 0 Files changed 13 +1,383 -1,361

NicolasHug commented on Feb 16, 2023 edited by pytorch-bot

This PR moves `functional_pil.py` into `_functional_pil.py` and let `functional_pil.py` have the same content (ish), with a warning. The changes are made for `functional_tensor.py`. These changes are hard to test because if we want to make sure all scenarios raise the proper warning, we need multiple executions of the Python interpreter. This can be done via `utils like this one` form scikit-learn which is used `like this`. ... can't be arsed to go through that right now. If you have a better idea, LMK :) As manual check, all those blocks below properly raise the warning:

```
from torchvision.transforms import functional_pil # warns
import torchvision.transforms.functional_pil as fp # warns

from torchvision.transforms.functional_pil import resize # warns
print(resize)

# Importing stuff that was already private breaks: this is for the best?
# from torchvision.transforms.functional_pil import _parse_fill # Breaks
# from torchvision.transforms.functional_tensor import _is_tensor_a_torch_image # Bi
```

Reviewers 2 more reviewers vfdev-5 pmeier

Assignees No one assigned

Labels cla signed code quality deprecation module: transforms

Projects None yet

Milestone No milestone

