CountAI - Submission Report

Mohamed Mafaz - IIT Madras: AM25M009

Data Preprocessing

Dataset Structure:

Images are stored in the Dataset directory, named as 0.jpg, 1.jpg, ..., etc.

Custom Dataset:

The Image_dataset class in Dataset.py loads images and assigns labels:

Images with filenames <50.jpg are labeled "Not Defective".

Images with filenames >=50.jpg are labeled "Defective".

Transforms:

Each image is:

Resized to (256, 256)

Randomly horizontally flipped

Randomly rotated (±10 degrees)

Color jittered (brightness, contrast, saturation)

Converted to a NumPy array for model input

Model Architecture

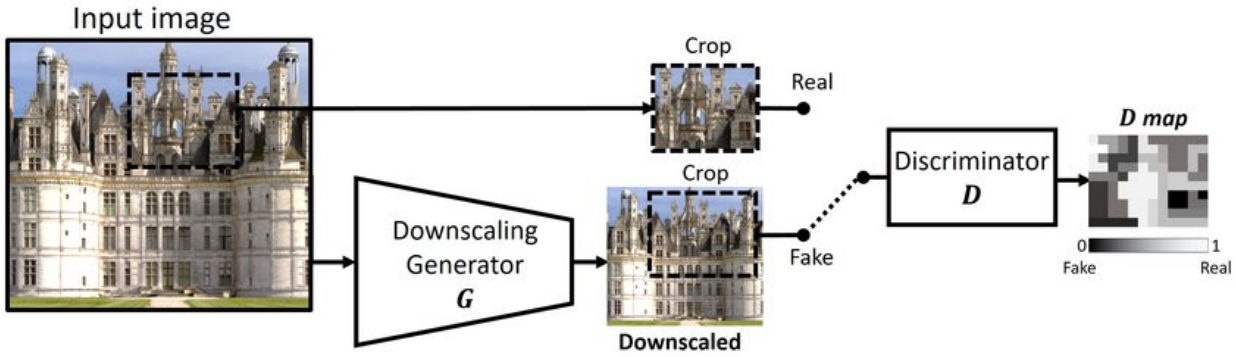
Discriminator Model:

Implemented as Discriminator in Critic.py, based on a convolutional neural network (CNN) with:

Initial convolutional block

Multiple ConvBlock layers (Conv2d + BatchNorm + LeakyReLU)

Final Conv2d layer for output



Patch-based Discriminator, first introduced in the pix2pix paper (Isola et al., 2017).

- Instead of the discriminator classifying an entire image as real or fake, it classifies small patches (like 30×30 pixels) independently.

- The final output is a grid of predictions (real/fake per patch), not just a single scalar.

Why Patch-based?

1.Local structure matters

Many image-to-image tasks (denoising, super-resolution, defect detection) don't need to check global realism — it's enough that each patch looks realistic.

2.Fewer parameters, faster training

Because the discriminator is only looking at local windows.

3.Encourages sharper details

Global discriminators can let blurry images pass. PatchGAN forces the generator to get local textures right.

Purpose:

The model acts as a binary classifier to distinguish between defective and non-defective images.

Training Methods

Data Loading:

Uses WeightedRandomSampler to balance classes during training (see train.ipynb).

Data is loaded in batches using DataLoader.

Loss Function:

Binary Cross Entropy with Logits (nn.BCEWithLogitsLoss)

Optimizer:

Adam optimizer with learning rate and betas set in the config.

Training Loop:

For each epoch, batches are processed:

Images are permuted to channel-first format.

Model outputs are squeezed and averaged.

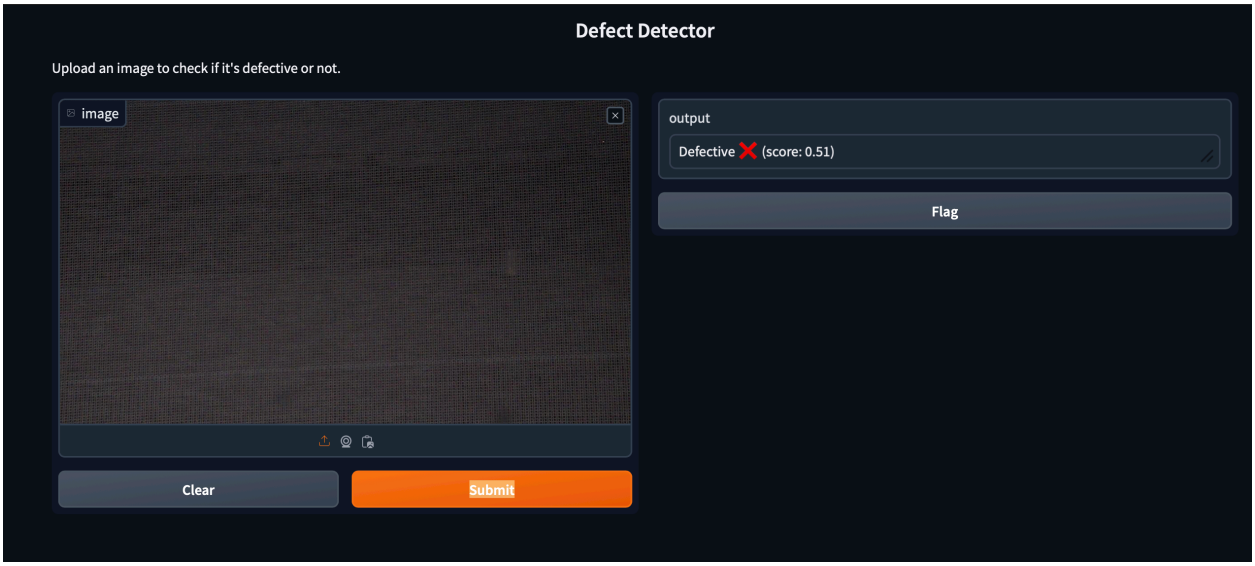
Labels are converted to binary targets.

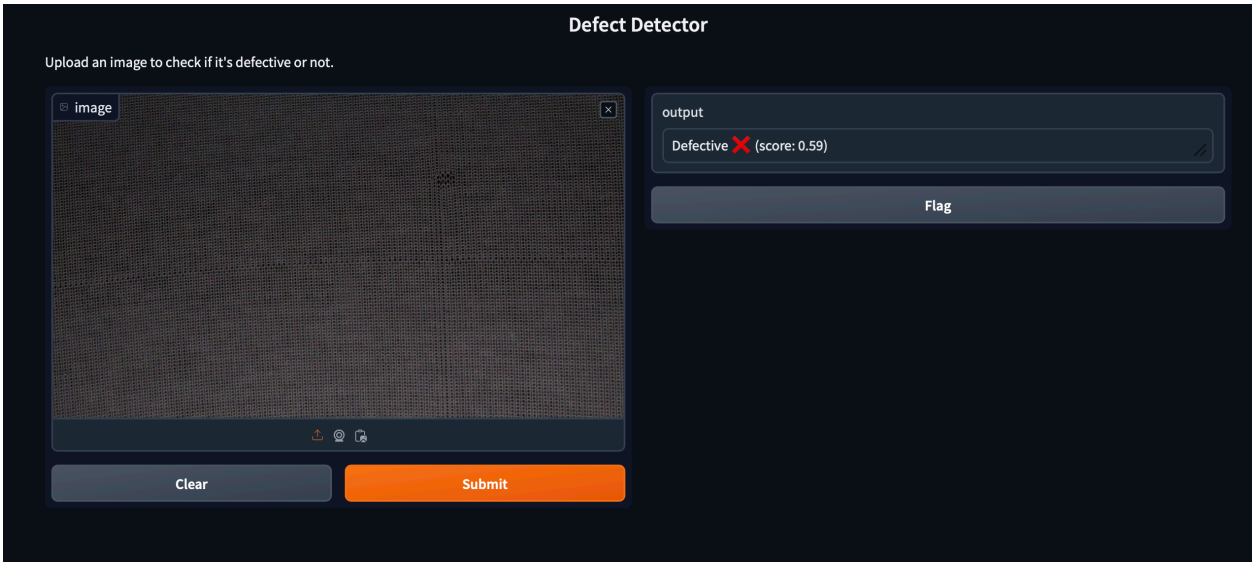
Loss is computed and backpropagated.

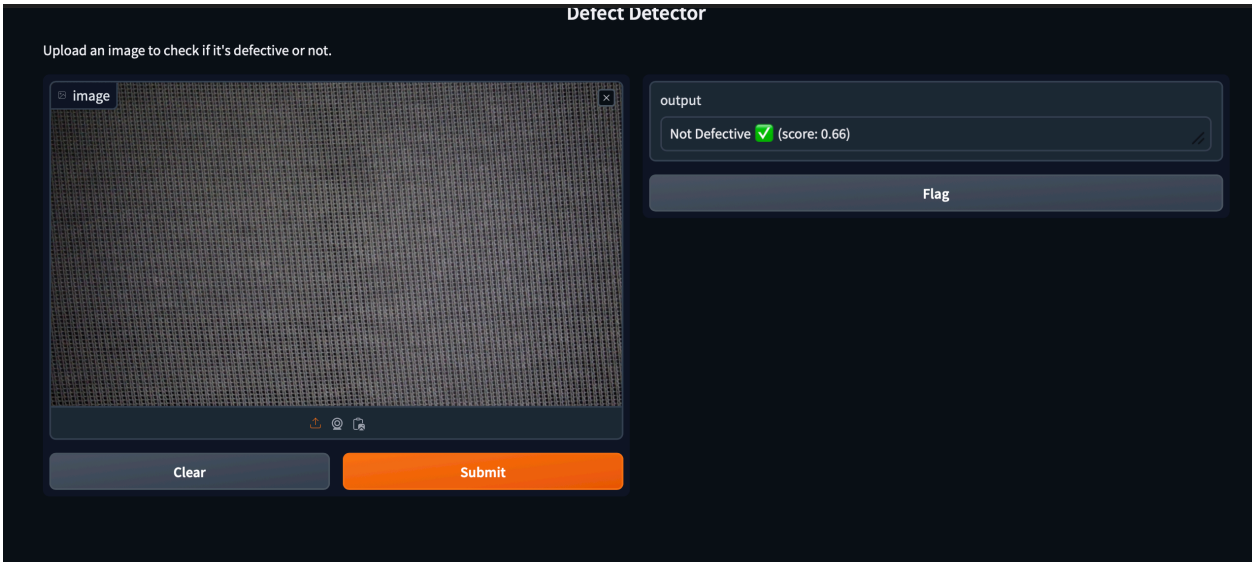
Model parameters are updated.

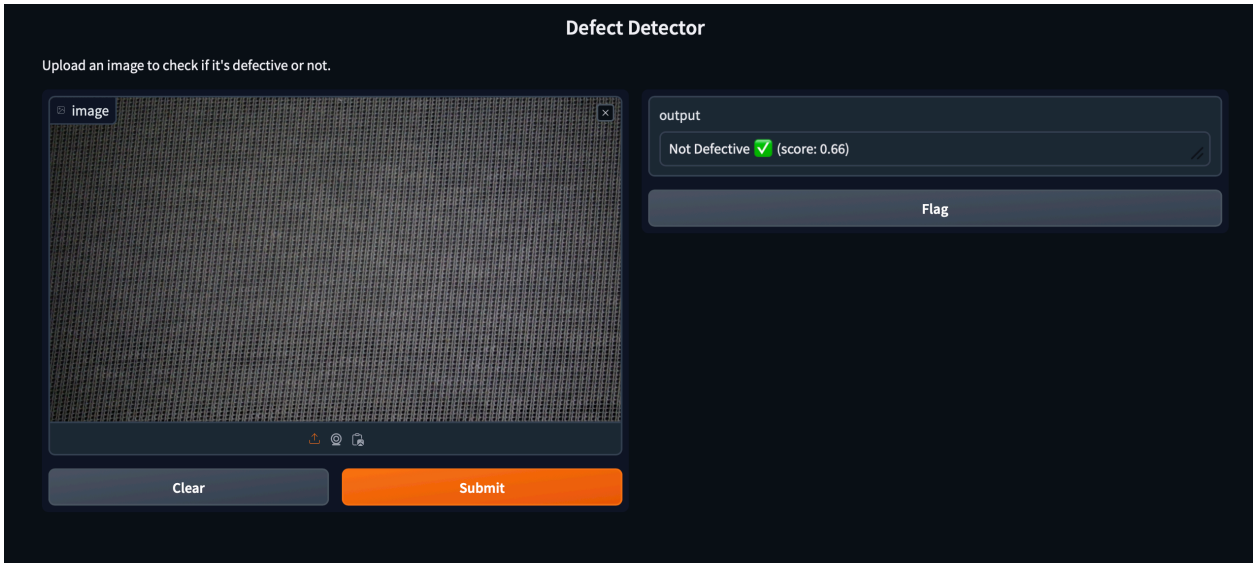
Model state is saved as critic.pth.

Deployment


Defect Detector
Upload an image to check if it's defective or not.
image
output
Defective (score: 0.51)
Flag
Clear
Submit


Defect Detector
Upload an image to check if it's defective or not.
image
output
Defective (score: 0.59)
Flag
Clear
Submit


Defect Detector
Upload an image to check if it's defective or not.
image
output
Not Defective (score: 0.66)
Flag
Clear
Submit



Gradio Interface:

The trained model is loaded in app.py and exposed via a Gradio web interface for user-friendly predictions.

Gradio via API:

```
from gradio_client import Client, handle_file
client = Client("Mafaz03/CountAI")
result = client.predict(
                image=handle_file('https://raw.githubusercontent.com/gradio-app/gradio/main/test/test_files/bus.png'),
                api_name="/predict")
```

Visit the page:

https://huggingface.co/spaces/Mafaz03/CountAI