Detection Training Approach

To enhance the model's robustness and prevent overfitting, an image from the dataset is selected and a random scale factor between $[1.6, 3)$ is applied. This ensures that the model encounters a variety of image scales during training, making it more adaptable and less likely to memorize specific images.

The scale factor is determined using a beta distribution, which is detailed below.

Why Beta Distribution?

The beta distribution is chosen to bias the selection towards lower scale values. This increases the frequency of slightly scaled images, making the model proficient at recognizing minor pixelation. Lower scale values result in images that are more similar to the original, posing a greater challenge for the model to detect pixelation, which enhances its detection capability.

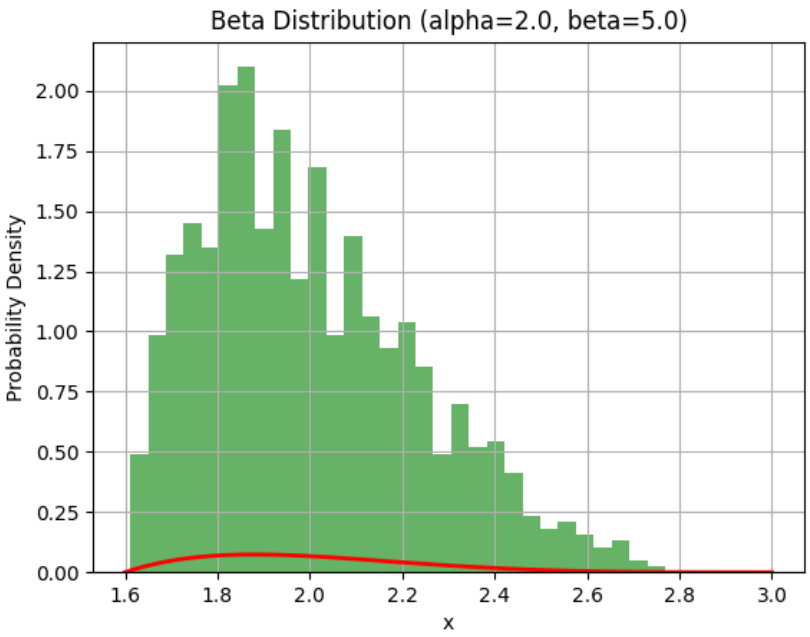

Figure 1: Beta Distribution Graph

Scale Factor Calculation

The following Python code snippet illustrates how to calculate the scale factor using a beta distribution:

```
alpha = 2.0
beta = 5.0

random_beta = np.random.beta(alpha, beta)
random_number = 1.6 + random_beta * (3 - 1.6)
scale_factor = round(random_number, 1)
```

Why the Range 1.6-3?

Through extensive analysis, documented in the 'pixel_detection_analysis' notebook, it was observed that pixelation becomes noticeably apparent at a scale factor of 2.0 or higher, regardless of the image resolution. Therefore, the range of 1.6 to 3 strikes a balance, ensuring exposure to both subtle and pronounced pixelation.

Major Advantage of This Training Method

This approach is resolution-agnostic, meaning it can effectively train models on images of any resolution, not just the 1920x1080 resolution. By focusing on scale factors, the method ensures consistent performance across different image sizes.

Two Methods Proposed for Efficient and Fast Detection

1) MobileNetV3_Small

Why this choice? MobileNetV3_small is an ideal choice due to its compact size, fast processing speed, and suitability for deployment on embedded systems.

2) Canny Edge Detection + MobileNetV3_Small (Novel Method)

This method combines Canny edge detection for pre-processing with MobileNetV3_small for detection.

Canny Edge Detection: (1)

Why this choice? This combined approach is not only faster but also more accurate than using MobileNetV3_small alone. This model not only performs better in this scenario where our task is to detect pixelated images it also performs better than the previous model for detecting images with mosaic blur as we'll discuss in the last section .By leveraging edge detection, the model can better identify pixelation patterns, improving overall detection accuracy.

Metrics

Method 1: MobileNetV3_Small

- Precision: 0.9893
- Recall: 0.9317
- F1 Score: 0.9597
- Test Accuracy: 0.961
- Test Loss: 0.1265
- Speed: 242086 FPS
- Model size: 5.844 MB

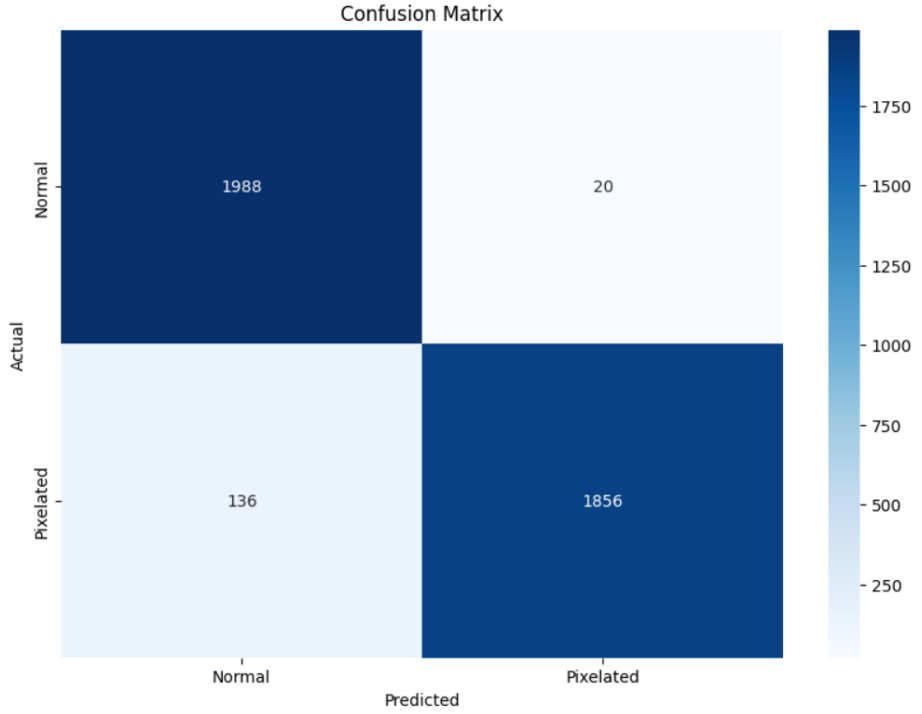

Figure 2: Confusion Matrix for Method 1

Method 2: Canny Edge Detection + MobileNetV3_Small

- Precision: 0.9920

- Recall: 0.9832
- F1 Score: 0.9876
- Test Accuracy: 0.9875
- Test Loss: 0.0357
- Speed: 270567 FPS
- Model size: 5.844 MB

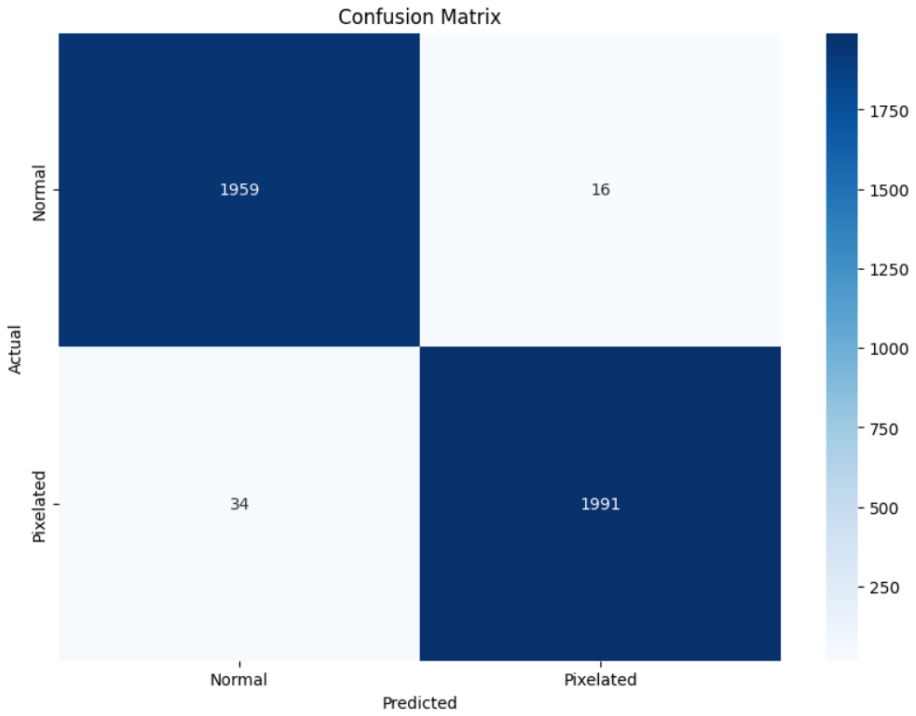

Figure 3: Confusion Matrix for Method 2

Unexpected Finding: Beyond Pixelation Detection

While the primary goal was to detect and fix pixelated images, it was discovered that the Canny edge detection method combined with MobileNetV3_small excels at detecting mosaic blur images as well. This unexpected finding suggests potential applications beyond the initial scope.

Proposed Solution for Fixing Mosaic Blur

Using a pix2pix model with a MobileNet backbone architecture is proposed. This approach shows promise in addressing mosaic blur, further enhancing the model's versatility.