Term Project Presentation (II)
Preprocessing Method for DCT-Based Image-Compression

Munenori Oizumi, Member, IEEE 2006

Presenter : 孔祥庭, 王翊丞

Outline

I. Introduction

II. Related Work

III. Methodology

- Design & Implementation
- Code segments

IV. Experiment

V. Reference

I. Introduction

- In DCT-based image compression, such as JPEG, visible ringing artifacts often appear at low bit rates. A straightforward way to reduce this distortion is to apply a global low-pass filter in the preprocessing stage.

- However, this approach inevitably causes the entire image to become blurred. The authors of this paper observed that **ringing artifacts mainly occur around areas with strong edges, rather than in smooth regions**.

I. Introduction

- To address this problem, they proposed a **selective preprocessing method** that calculates the **auto-correlation coefficient** for every pixel. When the coefficient exceeds a certain threshold, a low-pass filter is applied and its result is blended with the original image.

- In this way, only edge regions are filtered, while smooth areas remain untouched. This selective filtering effectively suppresses **ringing artifacts** while preserving overall image sharpness.

Outline

I. Introduction

II. Related Work

III. Methodology

- Design & Implementation
- Code segments

IV. Experiment

V. Reference

II. Related Work

➢ Pre-Processing Methods

Some methods attempt to **filter the input image before compression** to avoid producing severe artifacts. However, **simple linear filtering** often causes loss of fine detail and over-smoothing.

➢ Post-ProcessingMethods

Many researchers proposed **artifact removal filters** applied **after decompression**. However, they **require modifications on the decoder side**, which is not compatible with existing hardware or JPEG/MPEG standards.

Outline

I. Introduction

II. Related Work

III. Methodology

- Design & Implementation
- Code segments

IV. Experiment

V. Reference

III. Design & Implementation

- First, the algorithm analyzes local image characteristics using the **auto-correlation function**, which reflects the similarity between neighboring pixels.

- From this, a modified auto-correlation coefficient, ρ_{mod}, is calculated for each pixel.

- A large positive value of ρ_{mod} indicates a region with strong edges and a high likelihood of ringing distortion, while a negative value corresponds to high-frequency texture regions where ringing is less visible.

III. Design & Implementation

- If ρ_{mod} exceeds a given threshold, a low-pass filter is applied, the filtered result is weighted and combined with the original image.

- This selective filtering process is repeated in both the horizontal and vertical directions.

- As a result, **only edge regions are smoothed to suppress ringing artifacts**, while detailed textures and smooth areas are preserved.

III. Code segments

```python
def calculate_rho_mod(self, pixels):
    mean = np.mean(pixels)
    centered = pixels - mean
    R0 = self.calculate_autocorrelation(centered, 0)
    R1 = self.calculate_autocorrelation(centered, 1)
    if R0 + self.delta < 1e-10:
        return 0
    return R1 / (R0 + self.delta)
```

From the paper:

$$\rho_{\text{mod}} = R_{xx}(1)/(R_{xx}(0)+\delta)$$

Calculate the modified auto-correlation coefficient ρ_{mod}

```python
for y in range(height):
    for x in range(width):
        start_x = max(0, x - half_window)
        end_x = min(width, x + half_window + 1)
        local_pixels = image[y, start_x:end_x].astype(np.float64)
        rho_mod = self.calculate_rho_mod(local_pixels)
        rho_map_dir[y, x] = rho_mod
        if rho_mod > self.rho_threshold:
            filtered_pixels = self.apply_stronger_lowpass_filter(local_pixels)
            center_idx = x - start_x
            if center_idx < len(filtered_pixels):
                original_value = image[y, x]
                filtered_value = filtered_pixels[center_idx]
                intensity = min(1.0, (rho_mod - self.rho_threshold) * self.filter_intensity)
                filter_map[y, x] = intensity
                processed[y, x] = original_value * (1 - intensity) + filtered_value * intensity
```

If ρ_{mod} exceeds the threshold, a low-pass filter is applied to the local region. The filtered value is then **combined with the original pixel** using a weighting coefficient α derived from ρ_{mod}.

III. Code segments

```python
def preprocess_direction(self, image, direction='horizontal'):
    if direction == 'vertical':
        image = image.T
    H, W = image.shape
    processed = np.copy(image).astype(np.float64)
    filter_map = np.zeros((H, W))
    rho_map_dir = np.zeros((H, W))
    half = self.window_size // 2

    if direction == 'vertical':
        processed = processed.T
        filter_map = filter_map.T
        rho_map_dir = rho_map_dir.T

    self.filter_map = filter_map
    if direction == 'horizontal':
        self.rho_map = rho_map_dir
    return processed
```

- The function preprocess_direction() performs preprocessing in one direction at a time.
- When the direction is set to horizontal, it processes each row.
- When set to vertical, the image is transposed, allowing the same code to be reused.
- After both passes, ringing artifacts are suppressed in both orientations.

III. Code segments – Test conditions

```
compressor = DCTCompressor(quality=10)

uniform_preprocessor = UniformPreprocessor(filter_strength=1.0)

fixed_preprocessor = DCTPreprocessorFixed(
        rho_threshold=0.3,
        filter_intensity=2.5,
        window_size=9,
        delta=10
)

adaptive_preprocessor = DCTPreprocessorAdaptive(
        base_rho_threshold=0.3,
        adaptive_threshold=True,
        adaptive_range_variance=(500, 5000), #threshold modify
        adaptive_range_threshold=(0.2, 0.45),
        filter_intensity=2.5,
        window_size=9,
        delta=10
)
```

Compression quality

Global filtering

Fixed Threshold

Adaptive Threshold

III. Code segments - Adaptive Threshold

```
adaptive_preprocessor = DCTPreprocessorAdaptive(
        base_rho_threshold=0.3,
        adaptive_threshold=True,
        adaptive_range_variance=(500, 5000), #threshold modify
        adaptive_range_threshold=(0.2, 0.45),
        filter_intensity=2.5,
        window_size=9,
        delta=10
)
```

```
def _calculate_global_variance(self, image_y):
        return np.var(image_y)
```

```
if self.adaptive_threshold:
        variance = self._calculate_global_variance(Y)
        v_min, v_max = self.adaptive_range_variance
        t_min, t_max = self.adaptive_range_threshold
        self.current_rho_threshold = np.interp(variance, [v_min, v_max], [t_min, t_max])
```

We calculate the Y channel. This single value represents the image's overall contrast.

- **High Variance:** A complex, high-contrast image (many details and textures).
- **Low Variance:** A simple, low-contrast image (many flat areas).

We use np.interp to create a dynamic mapping:

- If variance is **high**: Use a **high threshold** (0.45). This makes the filter to protect real textures.
- If variance is **low**: Use a **low threshold** (0.2). This makes the filter to reduce noise in flat area.

Outline

I. Introduction

II. Related Work

III. Methodology

- Design & Implementation
- Code segments

IV. Experiment

V. Reference

IV. Experiment

➢ Dataset

https://jpegai.github.io/test_images/

IV. Experiment

1. Test Conditions

Compression Quality = 10

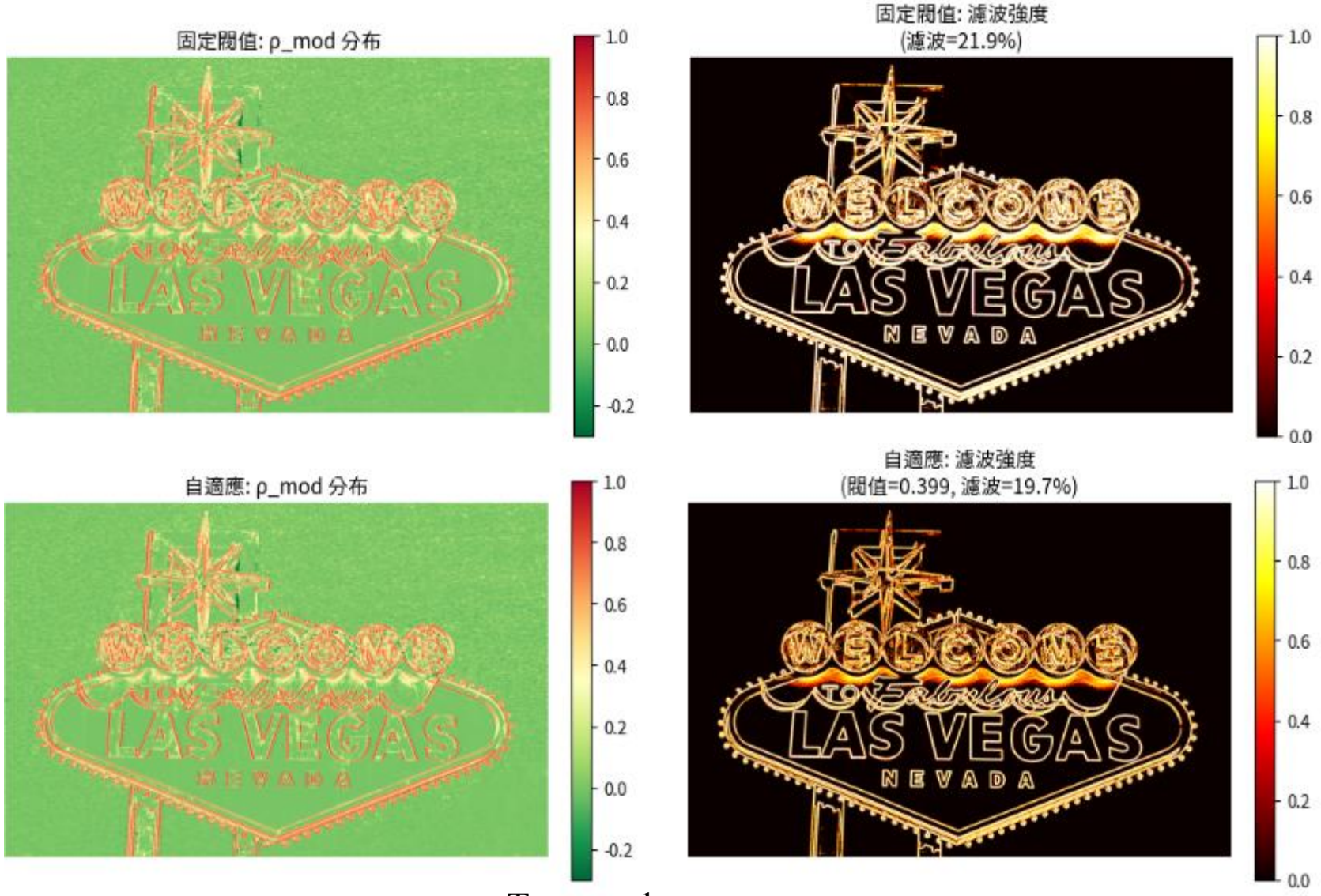
1. **Global filtering intensity = 1.0**

2. **Fixed threshold**
   - **rho_threshold=0.3**
   - **window_size=9**

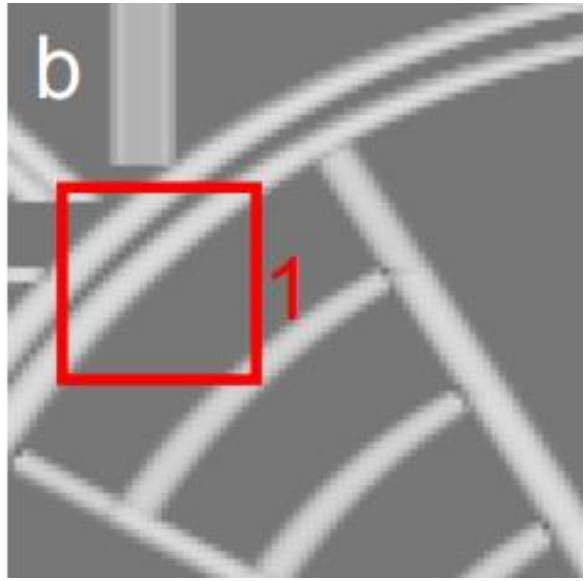
3. **Adaptive threshold**
   - **threshold range = (0.2, 0.45)**
   - **window_size=9**

https://jpegai.github.io/test_images/

IV. Experiment



Test result



Distribution of ρ_{mod}

IV. Experiment

Original image

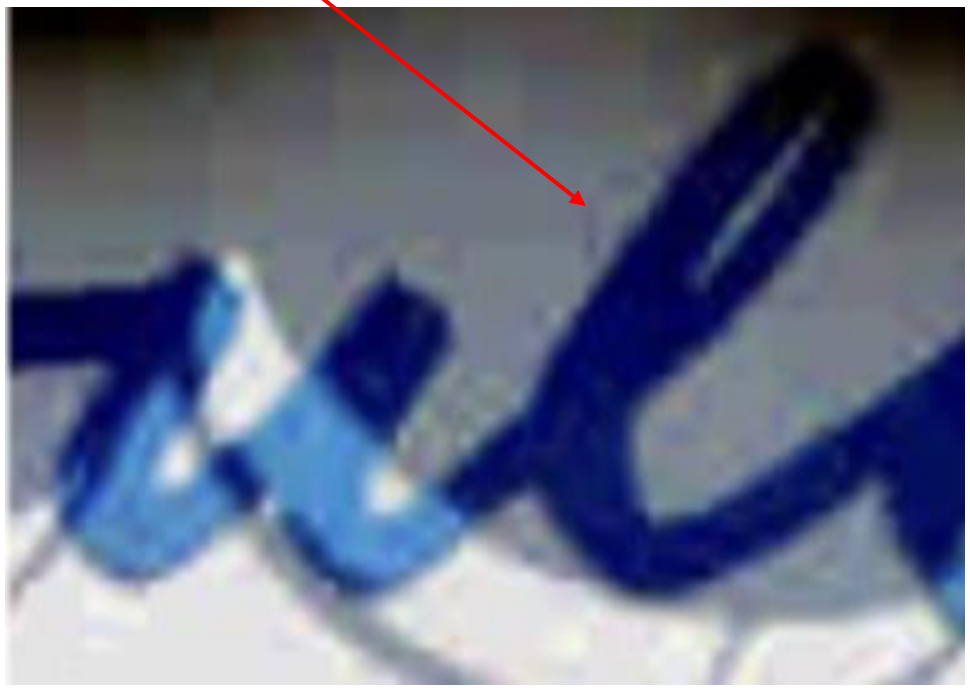
Only Compression

Compression with preprocessing
Adaptive Threshold

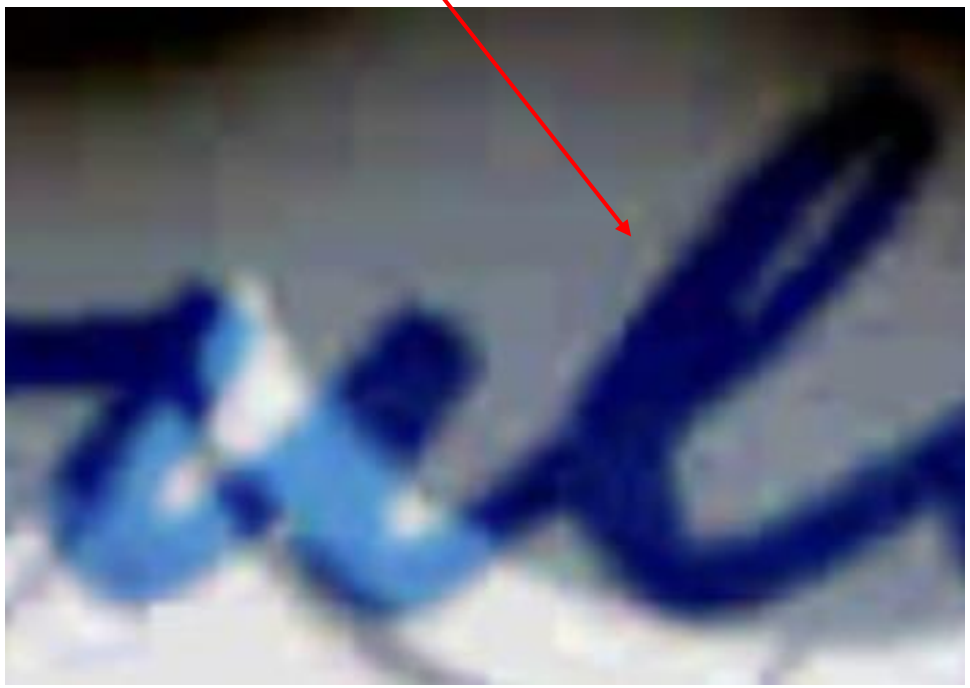
https://jpegai.github.io/test_images/

IV. Experiment

- As we can see that this method successfully reduce the ringing effect



Only Compression

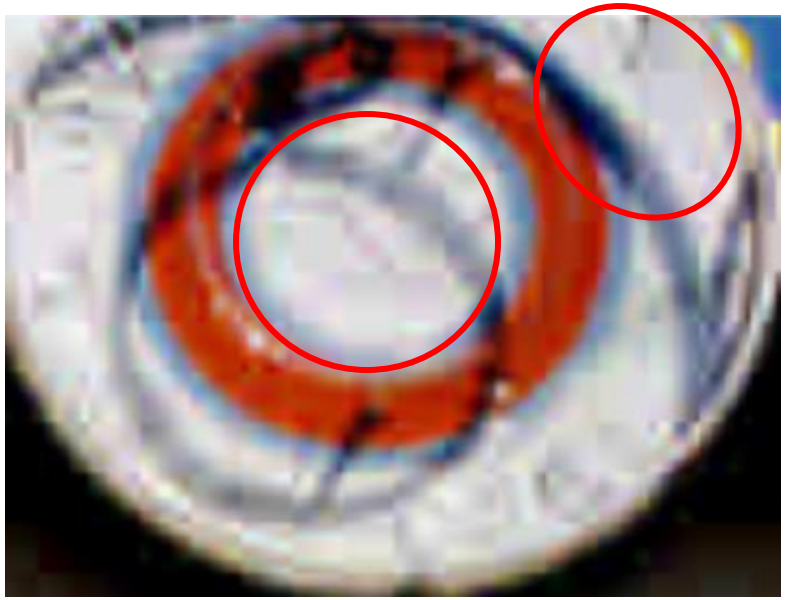


Compression with preprocessing
Adaptive Threshold

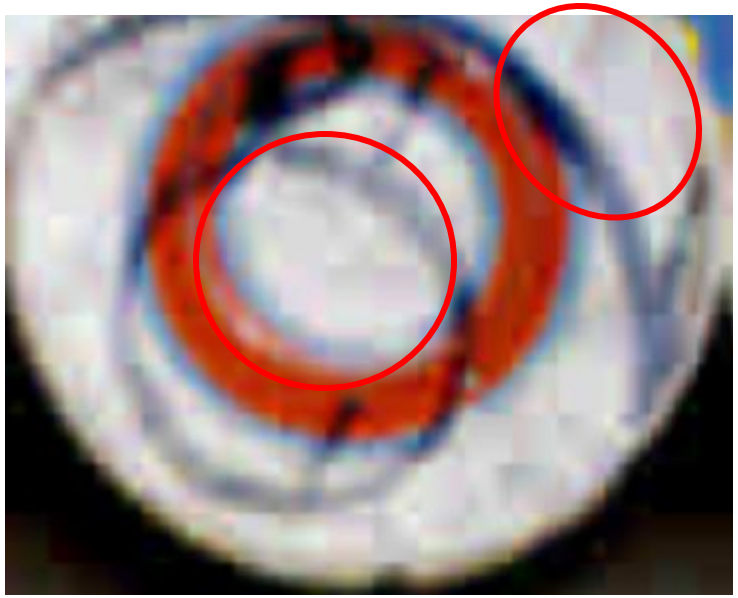
IV. Experiment

- As we can see that global filtering will cause all images to become blurred

Original image



Compression with preprocessing
Adaptive Threshold



Global Filtering

https://jpegai.github.io/test_images/

IV. Experiment

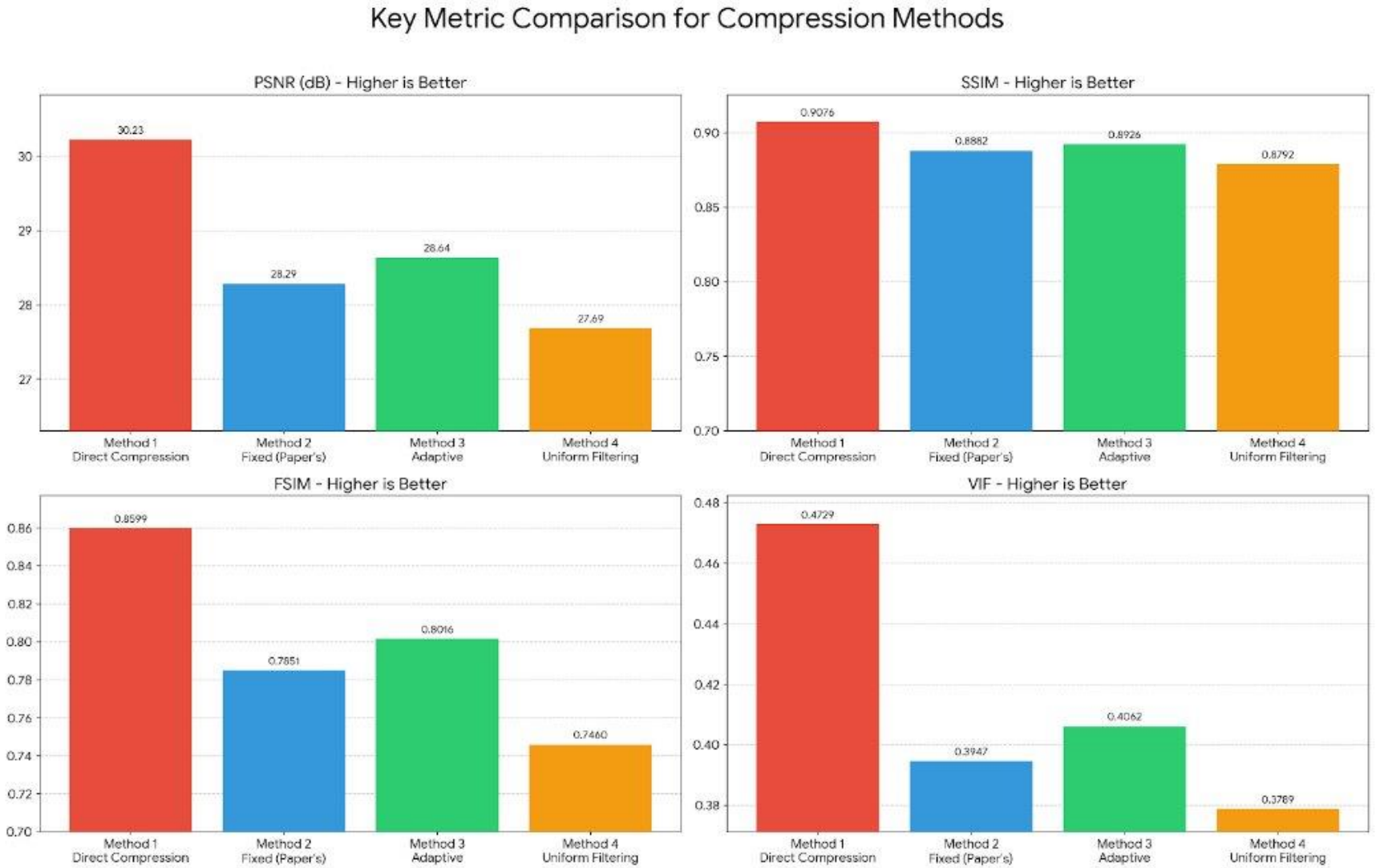
Direct Compression PSNR: 30.23 dB | SSIM: 0.9076 | FSIM: 0.8599 | VIF: 0.4729

Fixed Threshold Preprocessing (Paper's Method): PSNR: 28.29 dB | SSIM: 0.8882 | FSIM: 0.7851 | VIF: 0.3947

Adaptive Threshold Preprocessing: PSNR: 28.64 dB | SSIM: 0.8926 | FSIM: 0.8016 | VIF: 0.4062

Method 4: Uniform Filtering Preprocessing PSNR: 27.69 dB | SSIM: 0.8792 | FSIM: 0.7460 | VIF: 0.3789

IV. Experiment



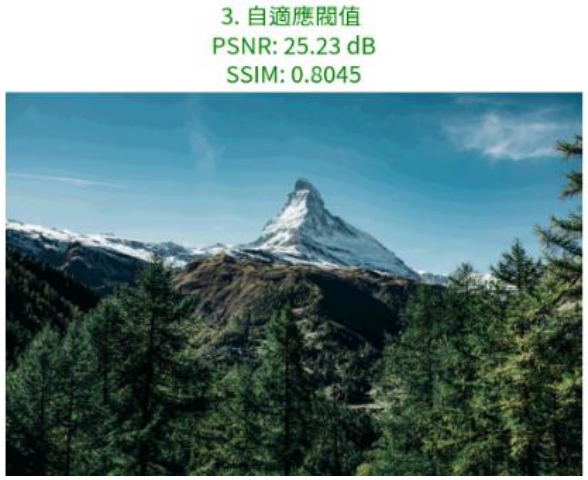
IV. Experiment

Original image

Adaptive Threshold

Global Filtering

IV. Experiment



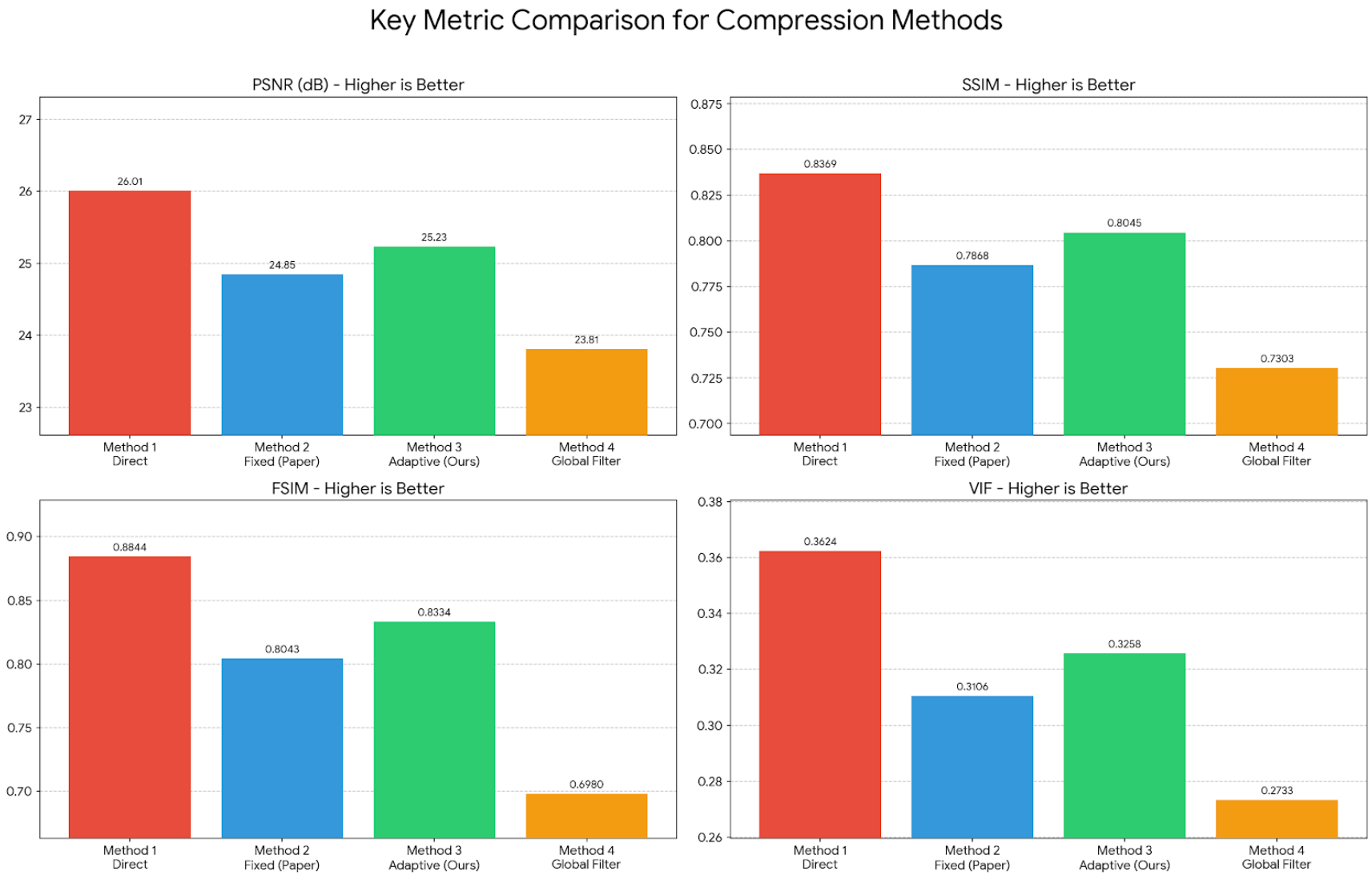
Direct Compression PSNR: 26.01 dB | SSIM: 0.8369 | FSIM: 0.8844 | VIF: 0.3624

Fixed Threshold Preprocessing (Paper's Method) PSNR: 24.85 dB | SSIM: 0.7868 | FSIM: 0.8043 | VIF: 0.3106

Adaptive Threshold Preprocessing (Optimized Method) PSNR: 25.23 dB | SSIM: 0.8045 | FSIM: 0.8334 | VIF: 0.3258

Global Filtering Preprocessing PSNR: 23.81 dB | SSIM: 0.7303 | FSIM: 0.6980 | VIF: 0.2733

IV. Experiment



V. Reference

[1] Oizumi, M. (2006). "Preprocessing Method for DCT-Based Image-Compression". *IEEE Transactions on Consumer Electronics*.

[2] Zhang, L., et al. (2011). "FSIM: A Feature Similarity Index for Image Quality Assessment".

[3] Sheikh, H. R., & Bovik, A. C. (2006). "Image Information and Visual Quality".

[4] Python Libraries: NumPy, SciPy, scikit-image, piq, Matplotlib.