Deepshield 🛡️

✨ Introduction

Deepshield is a robust watermarking project that demonstrates how to embed and extract digital watermarks in images using advanced signal processing techniques:

- **Discrete Wavelet Transform (DWT)**
- **Discrete Cosine Transform (DCT)**

The watermarking is performed in the frequency domain to ensure resistance to common image manipulations.

⚙️ Installation

1. **Install Poetry** (if not already installed):
   ```bash
   curl -sSL https://install.python-poetry.org | python3 -
   ```

2. **Navigate to the project directory:**
   ```bash
   cd deepshield_watermark
   ```

3. **Install dependencies:**
   ```bash
   poetry install
   ```

4. **Activate the Poetry environment:**
   ```bash
   poetry env activate
   ```

▶️ Usage

🔧 Running the Core Watermarking Module

To run the main watermarking and extraction logic:
```bash
python3 -m main
```

🔷 Running the Demo Notebook

🚀 Demo Options

You can explore the watermarking flow using either a Jupyter Notebook or a Python script, depending on your preference:

🧪 Option 1: Interactive Demo (Notebook)

File: `demo.ipynb`
Ideal if you prefer a step-by-step, visual walkthrough using Jupyter.

Steps performed in the notebook:
1. **Load** the original image.
2. **Apply watermark** and generate the watermarked image.
3. **Register** the watermarked image and associated watermark in a local/remote store.
4. **Load candidate image** for verification.
5. **Verify** the candidate image:
   - Retrieve potential matches
   - Perform image verification
   - Check semantic integrity

To run:
1. Open the notebook:
   ```bash
   jupyter notebook demo.ipynb
   ```
2. Follow the step-by-step execution of watermarking and verification pipeline.

🛠️ Option 2: Script Version

File: `demo_script.py`

Perfect for quick runs or integrating into automated pipelines.

```bash
   python demo_script.py
```
The script performs the same steps as the notebook but in a linear script format — no manual cell-by-cell execution required.

⚖️ Core Functionality

🖋️ Embedding Process

1. Perform two-level **DWT** to isolate low-frequency components.
2. Apply **zigzag scanning** to reorganize coefficients diagonally.
3. Transform diagonals using **DCT**.
4. Embed watermark bits (e.g., ±1) in diagonal frequency components.
5. Reconstruct the watermarked image via **IDCT → Inverse Zigzag → IDWT**.

🔎 Extraction Process

1. Apply **DWT + DCT** to the watermarked image.

2. Compute difference between frequency components to extract watermark bits.
3. Compare extracted watermark to original for **similarity evaluation**.

👜 Zigzag Reordering

- Enhances efficiency in frequency-domain embedding by traversing diagonally.

🔬 Mask and Similarity Score

- A **mask** identifies embedded watermark positions.
- A **similarity score** validates extraction accuracy.

🚀 Expected Output

- If implemented correctly, the **extracted watermark** should match the **original watermark** with near or complete accuracy.
- Tuning parameters like `alpha` can further enhance robustness and stability.

🎓 Authors and acknowledgment

* [Mohamed Njeh](Njeh@iabg.de) - Project Lead / Product Owner
* [Nehal Vaghasiya](Vaghasiya@iabg.de) - Developer
* [Paolo Notaro](notaro@iabg.de) - Developer
* [Olizier Charrez](cahrrez@iabg.de) - Developer