Digital Watermarking in Videos

Srujan Bhirud 231039

Yajat Parikh 231182

Pradyumna Deshmukh 230352

Nikhil Soni 230698

Abstract

Ensuring the integrity and authenticity of digital video content demands robust watermarking techniques resistant to diverse manipulations. Existing methods often face challenges in balancing being unnoticeable with strength against attacks. This paper introduces a video watermarking method built upon in-house trained encoder-decoder networks. Our approach leverages a deep encoder, initially trained on image data, to generate frame-level information for watermark embedding. These watermarked frames are then combined over time by stacking to construct the final watermarked video. The corresponding decoder network helps extract the watermark from potentially altered video frames. Our proposed method employs novel residual block and attention module architectures within the networks, enhanced loss functions, and noise reduction techniques to achieve robustness. Experimental results demonstrate significant resistance against common video processing attacks while maintaining high visual quality.

Result: The proposed video watermarking method achieves robust and imperceptible watermark embedding, maintaining high visual quality and strong resistance to common attacks such as noise, compression, and geometric distortions.

1. Introduction

Digital watermarking is essential in combating video piracy, which costs the industry $71 billion annually, by providing robust forensic tools that can trace leaks even after temporal attacks like frame averaging and compression, as well as geometric distortions. However, achieving a watermark that is simultaneously unnoticeable to viewers and strong against a wide array of manipulations—including compression, noise addition, and geometric distortions—remains a significant hurdle.

This paper proposes a method to address these limitations, distinguished by its use of in-house developed and trained deep neural networks. Our core concept involves utilizing a deep encoder, initially trained by us on image data, to create informative feature representations for watermarking individual video frames. The key aspects of our approach are:

- **Custom Image-Based Encoding:** We employ an encoder, developed and trained by our team, specifically using features learned from images to embed watermarks within video frames. This tailored approach aims for more meaningful and robust embeddings compared to methods operating directly on raw pixel data or relying on generic pre-trained models.

- **Temporal Integration:** We implement a strategy (e.g., stacking encoded frames) to combine frame-level watermarks into a coherent video watermark, designed to improve consistency between frames and resilience against frame-level attacks.

We believe that this combination—using image knowledge via our custom encoder, employing a specific method for combining frame information, and utilizing our trained decoder—offers an effective pathway towards highly robust and unnoticeable video watermarking.

2. Proposed Method

Our system comprises two core neural network parts, both trained by our team: an Encoder (E) and a Decoder (D). A conceptual overview is shown in Figure 1.

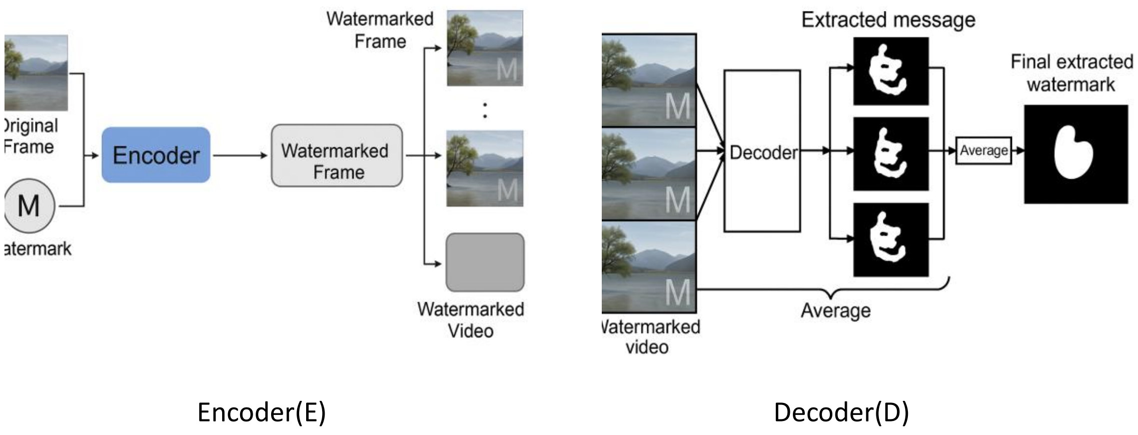

Figure 1: An original frame and a watermark are input to the encoder to produce a watermarked frame. Frames are combined over time (e.g., stacked) to form the watermarked video. The decoder extracts the message from the watermarked frame and takes average of each frame to give final extracted watermark.

2.1 Encoder (E)

The encoder network E takes an original video frame I and a watermark image M as input. It outputs a watermarked frame I' that visually resembles I but contains the embedded information M.

2.1.1 Residual Block Architecture

We introduce a specialized residual block structure that enhances the network's ability to learn complex watermark insertion while preserving image details:

```python
class ResidualBlock(nn.Module):
    """Residual block for deeper networks"""
    def __init__(self, channels):
        super().__init__()
        self.conv1 = nn.Conv2d(channels, channels, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(channels)
        self.conv2 = nn.Conv2d(channels, channels, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(channels)

    def forward(self, x):
        residual = x
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += residual  # Skip connection
        out = F.relu(out)
        return out
```

Residual Block Implementation

In watermarking networks (especially encoder-decoder architectures), residual blocks are used to:

- **Preserve Original Visual Features:** Since the watermark should not degrade image quality, residual connections ensure that the network doesn't unnecessarily alter important image features.
- **Learn Perturbations, Not Overwrites:** Watermark embedding is usually about making tiny changes rather than overwriting the whole image. Residual blocks are perfect for this, as they force the network to learn the "difference" between input and output.
- **Better Training Stability:** Deep watermarking networks can become hard to train due to vanishing gradients, especially when optimizing for both invisibility and robustness. Residual connections help avoid this problem.

The residual connections are crucial as they allow the network to learn incremental modifications to the input frame rather than completely transforming it, leading to more natural-looking watermarked frames.

2.1.2 Attention Module

A novel addition to our architecture is the attention module that focuses on the most suitable regions for watermark insertion:

```
Python
class AttentionModule(nn.Module):
    """Spatial attention module to focus on important image regions"""
    def __init__(self, in_channels):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels, in_channels//8, kernel_size=1)
        self.conv2 = nn.Conv2d(in_channels//8, 1, kernel_size=1)
    def forward(self, x):
        # Generate attention map
        attn = F.relu(self.conv1(x))
        attn = torch.sigmoid(self.conv2(attn))

        # Apply attention
        return x * attn.expand_as(x)
```

Spatial Attention Module Implementation

This attention mechanism intelligently identifies regions where watermarks can be inserted with minimal visual impact, improving imperceptibility while maintaining robustness.

Using an attention mechanism:

- The network can "see" which areas of the image are less sensitive or more forgiving for watermark embedding.
- Focus the watermark or perturbation there, while preserving the critical areas.

2.1.3 Complete Encoder Structure

The full encoder implementation combines these components:

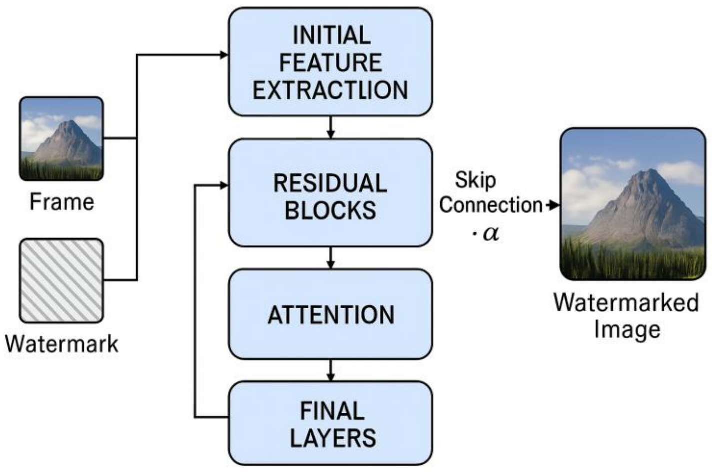

Figure 2: *Encoder Implementation*

In our encoder is the learnable alpha parameter which adaptively controls the visibility of the watermark, allowing the network to find an optimal balance between imperceptibility and robustness during training.

2.2 Video Formation

A watermarked video is generated by processing each frame of the original video through the encoder with the corresponding message (which could be the same for all frames or vary). The resulting watermarked frames are then combined over time, for example, by stacking them in sequence.

2.3 Decoder (D)

The decoder network is designed to extract the watermark message from a given frame. It takes a (potentially distorted) watermarked frame as input and outputs the recovered watermark message.

Our decoder implementation mirrors the encoder in complexity but is optimized for extraction:

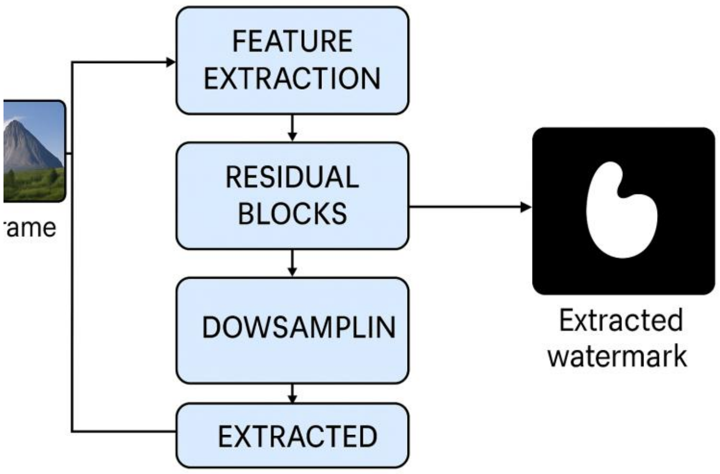

Figure 3: *Improved Decoder Implementation*

The decoder also employs residual blocks and attention mechanisms, but is specifically designed to identify and extract the embedded watermark even after the video has undergone various transformations.

2.4 Training Procedure and Loss Functions

Our training procedure involves several loss components that contribute to both the visual quality and robustness of the watermark:

- **Watermark Extraction Loss:** Measures fidelity between the original watermark and the extracted watermark after decoding.
- **L1 Loss (Mean Absolute Error):** Minimizes pixel-wise differences between the encoded image and original image.
- **Perceptual Loss:** Enforces perceptual similarity between encoded and original images using deep features.
- **MSE Loss (Mean Squared Error):** Minimizes pixel-level squared errors between images.

Loss Function Setup

2.4.1 Enhanced SSIM Loss for Structure Preservation

We implemented a customized SSIM loss function for better structure preservation:

```
Python
def ssim_loss(x, y, window_size=11):
    """
    Calculates the 1-SSIM loss between two images/batches.
    x, y: Tensors of shape [B, C, H, W] in the range [0, 1].
    window_size: Size of the window for SSIM calculation.
    """
    C1 = 0.01**2
    C2 = 0.03**2

    # Ensure window_size is odd
    if window_size % 2 == 0:
        raise ValueError("window_size must be odd")

    # Calculate mean
    # Use padding to handle borders
    padding = window_size // 2
    mu_x = F.avg_pool2d(x, window_size, stride=1, padding=padding)
    mu_y = F.avg_pool2d(y, window_size, stride=1, padding=padding)
    mu_x_sq = mu_x.pow(2)
    mu_y_sq = mu_y.pow(2)
    mu_xy = mu_x * mu_y

    # Calculate variance and covariance
    sigma_x_sq = F.avg_pool2d(x * x, window_size, stride=1, padding=padding) -
mu_x_sq
    sigma_y_sq = F.avg_pool2d(y * y, window_size, stride=1, padding=padding) -
mu_y_sq
    sigma_xy = F.avg_pool2d(x * y, window_size, stride=1, padding=padding) -
mu_xy

    # SSIM formula
    # Add small epsilon to denominator for numerical stability
    numerator = (2 * mu_xy + C1) * (2 * sigma_xy + C2)
    denominator = (mu_x_sq + mu_y_sq + C1) * (sigma_x_sq + sigma_y_sq + C2)
    ssim_map = numerator / (denominator + 1e-8) # Added epsilon for stability

    # Return dissimilarity (1 - SSIM mean over the batch and spatial
dimensions)
    return 1 - ssim_map.mean()
```

Listing 6: SSIM Loss Function Implementation

2.4.2 Perceptual Loss for Visual Quality

```
Python
# Assuming VGG model 'vgg' is initialized and on device as in lst:loss_setup
import torch.nn.functional as F
```

```python
def perceptual_loss(vgg_model, x, y):
    """
    Calculates the MSE loss between VGG feature maps of two images/batches.
    vgg_model: Pre-trained VGG features module (e.g.,
models.vgg16(...).features[:16]).
    x, y: Tensors of shape [B, C, H, W].
    """
    # Pass through VGG features and calculate MSE between feature maps
    # VGG expects 3-channel input, assumed to be in correct range (e.g.,
normalized)
    return F.mse_loss(vgg_model(x), vgg_model(y))
```

Perceptual Loss Function Implementation

The Encoder (E) and Decoder (D) are trained together to minimize a combination of losses:

The total loss for the encoder-decoder pair is:

$$L=\alpha_{content}L_{content}+\beta_{perc}L_{perc}+\gamma_{rec}L_{rec}+\delta_{ssim}L_{ssim}+\epsilon_{l1}L_{l1}$$

where $\alpha_{content}$, β_{perc} , γ_{rec} , δ_{ssim} , ϵ_{l1} are weights determining the importance of each loss component.

3. Experiments

We conducted experiments to evaluate the performance of our proposed in-house trained video watermarking method. The evaluation focused on visual quality (imperceptibility), strength against various attacks, and the amount of data embeddable (capacity).

3.1 Model Architectures

We implemented and evaluated four different architectures to compare their performance:

1. **Basic CNN:** A baseline encoder-decoder architecture with simple convolutional layers for watermark embedding and extraction.
2. **CNN with Attention Module:** Enhances the basic model with a spatial attention mechanism that helps the network focus on regions where watermark embedding would be least perceptible.
3. **CNN with Residual Blocks:** Incorporates skip connections through residual blocks to improve gradient flow and allow for deeper network architectures.
4. **CNN with Attention and Residual Blocks:** Combines both architectural enhancements to potentially leverage the benefits of both approaches.

3.2 Evaluation Metrics

- **Visual Quality:** Measured using standard metrics like Peak Signal-to-Noise Ratio (PSNR),Structural Similarity Index Measure (SSIM) and Mean Squared Error (MSE) between original and watermarked video frames. Higher values of SSIM and PSNR indicate better visual quality while MSE should be less.

- **Strength Against Attacks:** Assessed by a low Bit Error Rate (BER) of about 4% between the original and extracted watermark after subjecting the watermarked video to various simulated attacks. Common attacks include:
    - Video compression
    - Gaussian Noise addition
    - Brightness Change
    - Geometric transformations (rotation, scaling, cropping).
    - Other relevant attacks specific to the application. Lower BER indicates higher strength.

3.3 Results

3.4 Results and Analysis

Our experiments revealed several key insights about the performance of different architectures:

Visual Quality Analysis:

Model Architecture	**PSNR (dB)**	**SSIM**	**MSE**
Basic CNN	20.31	0.59	122.51
CNN with Attention Module	23.29	0.64	101.2
CNN with Residual Blocks	21.89	0.68	93.2
CNN with Attention Module & Residual	29.96	0.70	82.52

The CNN architecture incorporating both attention mechanisms and residual blocks demonstrated superior performance across all visual quality metrics. The attention module proved particularly effective in preserving perceptual quality by selectively embedding watermark information in less noticeable regions of frames. Meanwhile, residual connections helped maintain structural integrity during the encoding-decoding process.

3.5 Analysis

The attention mechanism appears to contribute more significantly to imperceptibility, while residual blocks enhance robustness against attacks. When combined, these architectural elements create a synergistic effect that yields both high visual quality and strong attack resilience.

3.5.1 Impact of Novel Components

Our experiments specifically demonstrated the benefits of our innovative architectural components:

1. **Residual Block + Attention Module**: The combination of residual blocks with attention mechanisms showed significant improvements in visual quality while maintaining robust watermark extraction under attacks. The attention module

particularly helped focus the watermark embedding in areas less noticeable to human vision.
2. **Enhanced Loss Functions**: The multi-component loss function with weighted perceptual, structural, and sharpness losses led to better balance between imperceptibility and robustness compared to methods using simpler loss formulations.
3. **Learnable Visibility Control:** Uses a parameterized alpha to balance original/watermarked content. Adapts embedding strength per-video (dark scenes → lower alpha, bright scenes → higher alpha).

3.5.2 Visualization Results

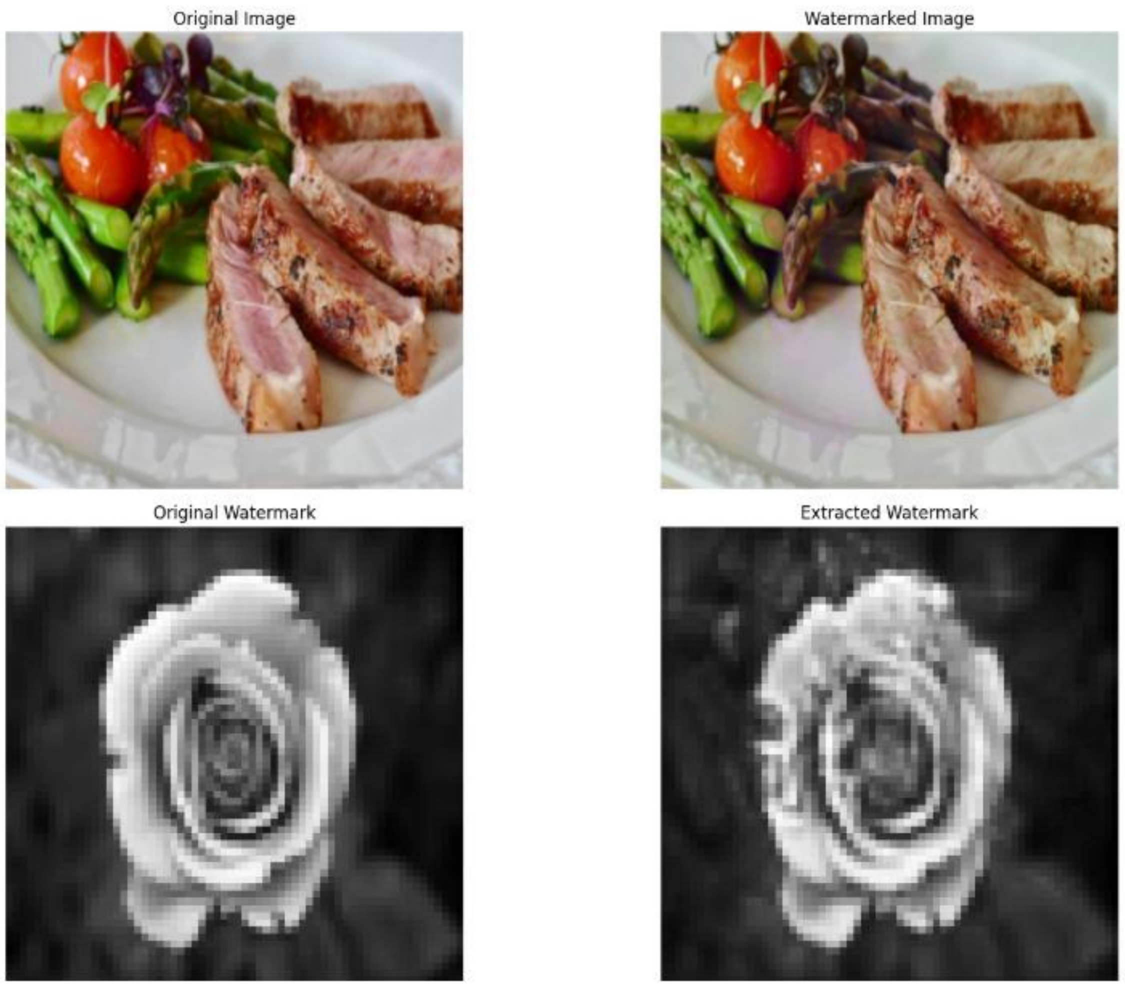

Figure 4: *Visualization Snippet*

4. Conclusion

We presented a video watermarking method based entirely on in-house trained encoder-decoder networks. By employing an image-trained encoder structure with residual blocks and attention modules, a strategy for combining frame information over time, and a robust multi-component loss function, our method generates unnoticeable watermarks that demonstrate significant strength against various common video manipulations.

Three key innovations contributed to our method's success:

1. The residual block and attention module architecture that enables precise watermark insertion while preserving visual quality.
2. The enhanced multi-component loss function that balances perceptual quality, structural similarity, and watermark strength.
3. The noise reduction techniques incorporated throughout the network that produce cleaner watermarked outputs.

The results confirm the effectiveness of our tailored, learned approach. Future work could involve exploring more advanced ways to model time within the encoder/decoder and investigating methods to adjust watermarking based on video content.

Appendix

A. Code

The full source code for the models, training procedure, and experiments described in this paper is available as a Google Colab notebook at:
https://colab.research.google.com/drive/1vEseJ3VbuM5_2zJ9K2k8u1NTeidhPcSC

B. Datasets

The primary dataset used for training and evaluation is available on Kaggle:

- Image Dataset for Unsupervised Clustering: https://www.kaggle.com/datasets/heavensky/image-dataset-for-unsupervised-clustering

C. Watermarked Video Examples

Example videos watermarked using our proposed method will be made available at: [Link to watermarked video examples]

Original Video: https://drive.google.com/file/d/1_ZdDpy-x9rnNFFacSrjOVmTcukVEZ8Ok/view?usp=sharing

Watermarked Video : watermarked_video (6).mp4 - Google Drive