Deinterlacing Based on BasicVSR

by

Chi Ruan

A thesis
submitted to the University of Ottawa
in partial fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Examining Committee

The following served on the Examining Committee for this thesis.

External Examiner:	Yuu Ono Professor, Department of Systems and Computer Engineering, Carleton University
Internal Member:	Robert Laganière Professor, School of Electrical Engineering and Computer Science, University of Ottawa
Supervisor(s):	Jiying Zhao Professor, School of Electrical Engineering and Computer Science, University of Ottawa

Declaration of Authorship

I hereby declare that this thesis is entirely my own work and has not been submitted for any other degree or qualification. All sources of information and assistance received during the course of this research have been properly acknowledged and cited. I have not plagiarized any part of the thesis, and all data and content presented are authentic and original.

Abstract

Deinterlacing is a critical technique in the field of digital video processing, aimed at converting interlaced videos into a progressive format. Interlaced video, a legacy from the analog television era, was developed to optimize bandwidth and storage constraints in earlier broadcast systems. It displays alternating lines of a frame in two separate fields, which can result in visual artifacts such as flickering and blurring when viewed on modern progressive displays. Due to the demand for superior video quality, deinterlacing technology has evolved, incorporating sophisticated algorithms and methodologies to address the artifacts and deficiencies inherent in interlaced video. In recent years, methods based on deep learning have achieved significant advancements.

Video super-resolution is another video restoration technique that aims to enhance the resolution of video frames by generating high-quality, detailed outputs from low-resolution inputs. This process focused on improving the clarity and sharpness of video content. While both deinterlacing and video super-resolution utilize temporal information to improve video quality, the latter has more extensive research and broader applications in the deep learning field.

By leveraging the advanced architectures and techniques developed for video super-resolution, deinterlacing models can be equipped with a robust framework for aggregating multiple misaligned frames. Based on a streamlined video super-resolution (VSR) baseline, BasicVSR, we propose a novel reconstruction technique for deinterlacing, and redesign the backbone network to prevent misalignment between different fields. As a result, we present a concise, efficient and versatile video deinterlacing baseline that achieves state-of-the-art performance, surpassing the second-best method by 1.18 dB on the Vimeo90K dataset. Additionally, we extend the model to include super-resolution functionality, enabling it to perform both deinterlacing and super-resolution tasks.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Jiying Zhao and co-supervisor Dr. Yu Liu, for their invaluable guidance, support, and encouragement throughout this research. Their expertise and insights have been instrumental in the completion of this work.

I would like to thank my team member, Jiaqi Yang, for his support and collaboration throughout this research. Your insights and encouragement have been greatly appreciated.

Dedication

To my parents, whose patience and understanding have been my guiding light throughout this journey. Your encouragement has inspired me to reach new heights.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Adam Adaptive Moment Estimation

CDV Candidate Direction Vector

CNN Convolutional Neural Network

CRT Cathode Ray Tube

DCN Deformable Convolutional Network

DLP Digital Light Processing

EDVR Enhanced Deep Video Restoration

EMA Exponential Moving Average

GAN Generative Adversarial Network

GD Gradient Descent

LCD Liquid Crystal Display

LSDV Lower Spatial Direction Vector

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MFDIN Multiframe Early Video Deinterlacing Network

MSE Mean Squared Error

NLP Natural Language Processing

NTSC National Television System Committee

PAL Phase Alternating Line

PDP Plasma Display Panel

PSNR Peak Signal-to-Noise Ratio

ReLU Rectified Linear Unit

ResNet Residual Network

RMSProp Root Mean Square Propagation

RNN Recurrent Neural Network

SECAM Sequential Color and Memory

SGD Stochastic Gradient Descent

SLA Spatial Line Average

SPyNet Spatial Pyramid Network

SSIM Structural Similarity Index

tanh Hyperbolic Tangent

TLA Temporal Line Average

USDV Upper Spatial Direction Vector

VDNet Video Deinterlacing Network

VSR Video Super-Resolution

Chapter 1

Introduction

Interlacing technology, as a technique to reduce bandwidth requirements, was widely used in broadcast television in the last century. Each interlaced video frame contains two fields captured in different moments. The first field includes all the odd lines of a frame, and the second field includes all the even lines. These fields are displayed alternatively, at a rate fast enough to create the illusion of a complete frame to the human's eyes. However, modern displays are mostly based on LCD technology, which is used to display progressive videos with all lines in each frame. The conflict between interlaced scans and modern displays is apparent. Deinterlacing is a process used to transform interlaced videos back to progressive formats.

In the following section, we introduce the background of deinterlacing in detail. Then, we describe the challenges and motivations, followed by an overview of our research on deep learning based deinterlacing. Finally, we summarize our contributions.

1.1 Background

Interlaced technology, a cornerstone in the history of television broadcasting, was developed to efficiently transmit video signals while minimizing bandwidth usage. The concept of interlaced video was first patented by a German engineer called Fritz Schröter [15] in 1930. Later, interlaced scans became the fundamental of analog television encoding systems, such as NTSC, PAL and SECAM. During that time, interlacing technology was prevalent in displays, such as CRT displays and ALiS plasma displays. This method effectively doubles the perceived frame rate without requiring double the bandwidth, a critical advantage in

the era of analog broadcasting. Interlacing helped to deliver smoother motion in video content and was instrumental in the widespread adoption of television.

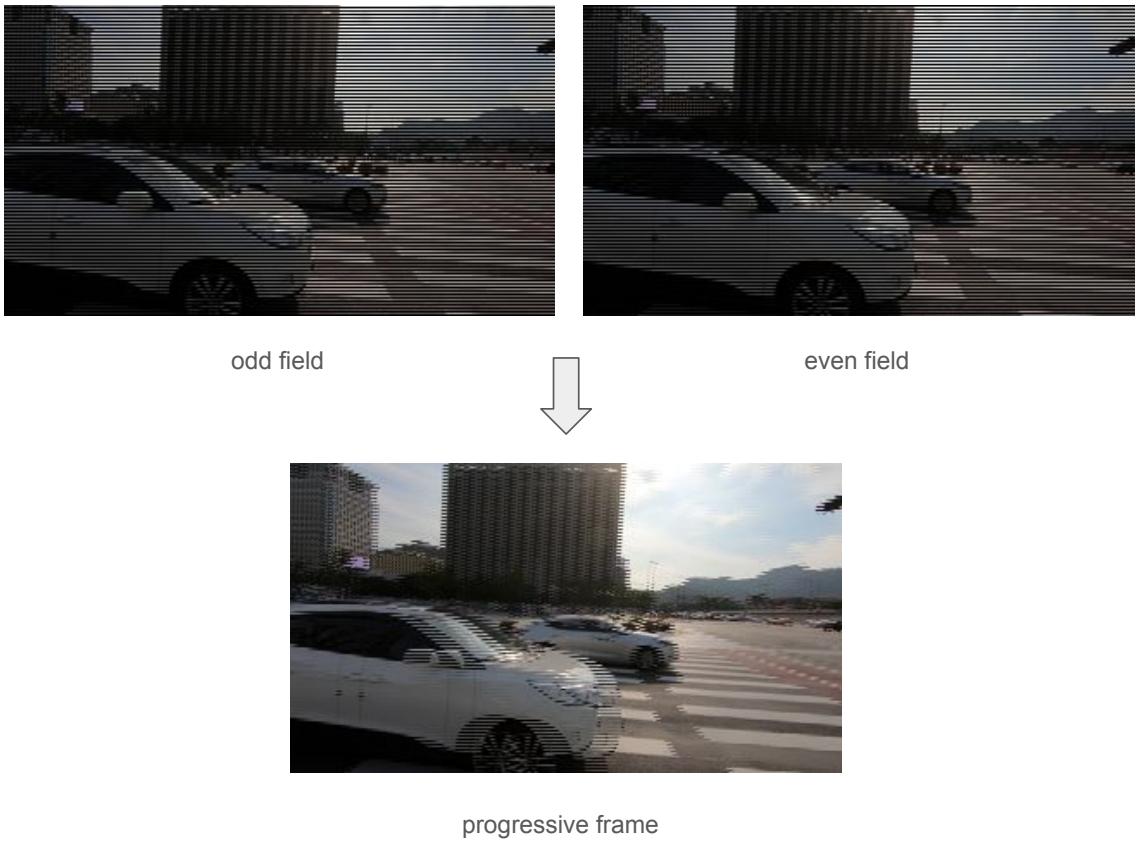


Figure 1.1: Odd field is generated by extracting all the odd lines of a frame in progressive format, and even field is generated by extracting all the even lines of the next frame. Then, the odd field and even field are combined together to generate the progressive frame. The frame shows noticeable interlacing artifacts, with combing visible along the edges of the car.

However, as modern displays, such as LCD, DLP and PDP, shifted towards progressive scanning, which displays all lines of a frame simultaneously, two interlaced fields must be combined as one complete frame. Since the two fields are taken at different moments, the drawbacks of combined video became pronounced. Issues such as flickering, combing, and motion artifacts become evident, particularly when objects in the videos are moving, shown in Fig. 1.1. There is a necessity for a solution to convert interlaced content into a progres-

sive format compatible with contemporary display technologies. Deinterlacing technology emerged as this solution, employing various algorithms and techniques to reconstruct the full frame from interlaced fields, thereby eliminating the artifacts and enhancing the visual quality of the video. By improving the clarity, reducing motion blur, and ensuring smoother playback, deinterlacing technology plays a crucial role in the integration of legacy video content into the modern digital landscape, providing viewers with a superior and more consistent viewing experience.

1.2 Challenges and Motivation

In the past few years, deinterlacing methods based on deep learning have emerged, achieving better results than conventional methods [6]. However, there are still several challenges to overcome.

1. **Combing effect:** Interlaced video is composed of two fields captured at different times, with each field containing half of the horizontal lines of a full frame. When these fields are deinterlaced into complete frames, the temporal disparity between them easily causes fast-moving objects or scenes to exhibit misaligned lines, resulting in a comb-like appearance. This temporal misalignment is particularly noticeable in areas with significant motion, where the offset between the fields creates visible gaps or serrations along the edges of moving objects.

2. **Temporal information aggregation:** Deinterlacing involves aggregating information from adjacent but misaligned frames. However, unlike other video restoration tasks, each field in an interlaced video includes half of the horizontal lines of a complete frame, and spatial disparity exists between neighbouring fields. When aggregating information from these misaligned fields directly, aliasing artifacts would occur. Especially the video contains a high level of detail in the horizontal direction.

3. **Reconstruction method:** Considerable variation exists in reconstruction methods employed across different papers. For example, in [11], the model reconstructs deinterlaced videos by predicting the missing fields and combining them with the input fields. On the other hand, in [10], the model employs vertical pixel shuffle, which divides a field feature map into two equal parts along the depth axis, with the second half representing the missing field feature map. After all, there is no standard and universal reconstruction method for deinterlacing deep learning networks.

4. **Complex networks:** When reviewing these deinterlacing models, significant distinctions exist, and some have high complexity. For instance, in VDNet [9], base interlaced images are generated by two traditional deinterlacing techniques, Spatial Line Average (SLA) and Temporal Line Average (TLA), and a coarse adaptive module before temporal aggregation. In MFDIN [10], DCN-based implicit alignment with a U-net-shaped offset network is used to align and fuse adjacent feature maps with the central frame. Gao *et al.* [11] employ Unet-like bidirectional propagation with flow-guided refinement blocks for aligning and propagating temporal information to produce missing fields. These network architectures have widely varying propagation and reconstruction approaches. Thus, video deinterlacing lacks a strong and straightforward deep learning baseline, causing several limitations, including increasing model complexity, absence of standardization, and difficulties in reproducing the technology and making innovations.

1.3 The Proposed Methods

Video super-resolution (VSR) is a technique aimed at enhancing the spatial resolution of video sequences, effectively transforming low-resolution videos into high-resolution ones, as illustrated in Fig. 1.2. Although deinterlacing and video super-resolution (VSR) have different objectives, they both involve extracting spatial and temporal features from the input video to reconstruct each frame. Compared to deinterlacing, VSR has attracted more attention and research efforts, as evidenced by more publications, conferences, and more demand in the real world. According to Chan *et al.*'s studies [16], there are four basic components in a VSR model: propagation, alignment, aggregation, and upsampling. For each component, their paper analyzed and compared different techniques from previous models [17–22]. With slight modifications of former options, the paper presented a simple and robust baseline called BasicVSR, which achieved a state-of-the-art performance in both quality and speed. Due to the similarity between deinterlacing and VSR tasks and the advancements in VSR research, we choose BasicVSR as our base model to introduce the efficiency and simplicity of the VSR method to the video deinterlacing task.

However, there are some incompatibilities between VSR and deinterlacing models. The reconstruction methods for super-resolution cannot be applied to deinterlacing. Most current deinterlacing models predict the missing field and combine the input and output fields [6, 7, 11, 23], which easily results in interlacing artifacts. Instead of generating a complete frame, these models focus on individual fields, which can compromise the quality of the final frame. There is a lack of a simple and practical reconstruction method for

Figure 1.2: Super-resolution aims to enhance the quality of low-resolution images on the left by reconstructing high-frequency details, resulting in clearer and more detailed visuals on the right.

deinterlacing. Inspired by pixel shuffle [24], we propose a novel technique called deinterlacing shuffle. The technique is a simple and efficient way to restore a feature vector by aggregating the information of surrounding feature vectors.

Moreover, in interlaced videos, the even and odd fields in the same frame contain different halves of the pixels, causing interlacing artifacts between neighbouring fields. Thus, when the model propagates information between fields in an interlaced format, errors will arise. To solve this problem, we add *Spatial Extraction* and *Reconstruction* before *Bidirectional Propagation* to utilize spatial information from each individual field and restore it into a full frame-size feature map before entering the propagation module to aggregate temporal information. By the invention of an original reconstruction method and redesigning of the BasicVSR [16] structure, this thesis presents a strong and simple video deinterlacing baseline, achieving state-of-the-art performance.

Thanks to the extensibility of the proposed model, we also introduce a super-resolution version that can not only transform interlaced videos to progressive format but also further increase their resolution. Interlacing technology was prevalent in the last century, and the videos produced at that time had limited resolution due to technological constraints. Therefore, the deinterlacing model incorporating super-resolution capability can address this common issue with one single process, which is time-saving and fully leverages all information from the interlaced video. It is the first model that can fulfill both deinterlacing and super-resolution tasks.

1.4 Contributions

This work makes the following contributions to the field:

- We propose a deinterlacing model based on BasicVSR, which achieves state-of-the-art results and introduces the advancement of VSR to the deinterlacing task.
- We introduce a novel deinterlacing reconstruction method called deinterlacing shuffle, which aggregates information from surrounding feature vectors to restore missing ones.
- We place the reconstruction module before propagation to restore fields into frames before aggregating temporal information, thereby avoiding misalignment between adjacent fields.
- We propose the first model capable of performing both deinterlacing and super-resolution tasks, which saves time and fully utilizes all the information in input videos.

1.5 Structure

The rest of this thesis is organized as follows.

- In Chapter 2, we provide background information on deep learning, covering its fundamental concepts, activation functions, common architectures such as CNNs, and key components of the training process.
- In Chapter 3, we conduct a comprehensive review of both traditional and deep learning-based deinterlacing methods, detailing their underlying principles while highlighting their strengths and weaknesses.
- In Chapter 4, we provide an in-depth explanation of our deinterlacing method, detailing the architecture of our deep learning model, its innovative features, and the rationale underlying our approach.
- In Chapter 5, we present a detailed description of our experiments and comparisons with other methods, including the specifics of the training process and comparative analysis in terms of PSNR and SSIM.

- In Chapter 6, we present the conclusions of the thesis, summarizing our contributions, and suggesting directions for future research.

Chapter 2

Background in Deep Learning

In this chapter, we introduce the fundamental concepts of deep learning, which include an overview of neural networks, an exploration of various deep learning architectures, and a discussion on training neural networks.

2.1 Neural Networks

A neural network is a computational model inspired by the human brain's structure and function. It consists of connected nodes or neurons, arranged in layers, as shown in Fig. 2.1. Signals, represented as numerical values, enter from the input layer, pass through one or more hidden layers, and are produced by the output layer. Inside each neuron, after it receives inputs from the previous layer, every input is multiplied by a corresponding weight learned during the training process. Then, all the products are summed together and go through an activation function, a non-linear calculation. Finally, the neuron sends the output to neurons in the next layer.

Neural networks are composed of multiple layers, each serving a specific purpose. The three main types of layers are the input, hidden, and output layers, as shown in Fig. 2.1. The input layer receives the initial data, then it is processed through one or more hidden layers. These hidden layers perform the core computations and feature extraction, transforming the input into a form that the output layer can use to make final predictions or classifications. The number and size of hidden layers can vary, depending on the complexity of the problem. Deeper networks with more layers are capable of learning more intricate patterns but may require more computational resources and careful tuning to avoid issues like overfitting.

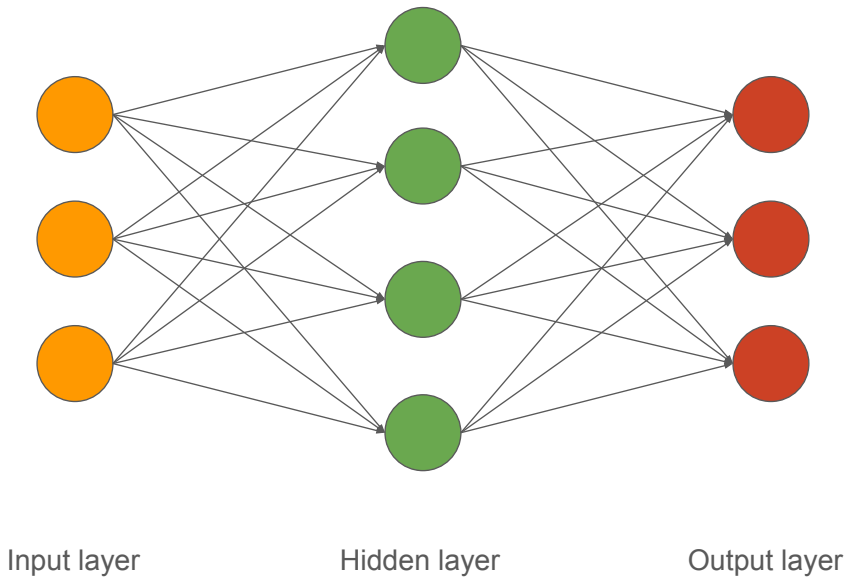


Figure 2.1: Neural network.

These networks are designed to recognize patterns and make decisions based on data inputs. By leveraging their layered architecture, neural networks can learn complex mappings from inputs to outputs, allowing them to generalize from examples and improve their performance over time. They excel at tasks that involve large amounts of data and intricate relationships, such as image classification, speech recognition, and natural language processing.

2.1.1 Weights and Biases

In a neural network, weights and biases are crucial parameters that are learned and optimized during the training process. They play a central role in determining the output of each neuron and, consequently, the network's ability to make accurate predictions or classifications.

Weights

Weights are numerical values assigned to the connections between neurons. Each connection between two neurons has a weight that signifies the strength or importance of that connection. During the forward pass of training, the input to a neuron is multiplied by its corresponding weight. These weights are adjusted through training using optimization algorithms like gradient descent, which minimizes the error between the predicted output and the actual target.

Mathematically, if a neuron receives inputs x_1, x_2, ..., x_n with corresponding weights w_1, w_2, ..., w_n, the weighted sum of the inputs is computed as:

$$z = \sum_{i=1}^{n} (x_i \cdot w_i). \tag{2.1}$$

Biases

Biases are additional parameters that allow the activation function to be shifted to the left or right, which helps the model to better fit the data. Each neuron has a bias value that is added to the weighted sum of the inputs before applying the activation function. The purpose of the bias is to provide flexibility in the decision boundary, enabling the network to model a wider range of functions. The equation incorporating the bias is:

$$z = \sum_{i=1}^{n} (x_i \cdot w_i) + b, \tag{2.2}$$

where b represents the bias term.

2.1.2 Activation Function

Activation functions play a crucial role in neural networks by introducing non-linearity into the model. This non-linearity enables the network to learn complex patterns and make accurate predictions. There are several common activation functions, and each has its own advantages and use cases.

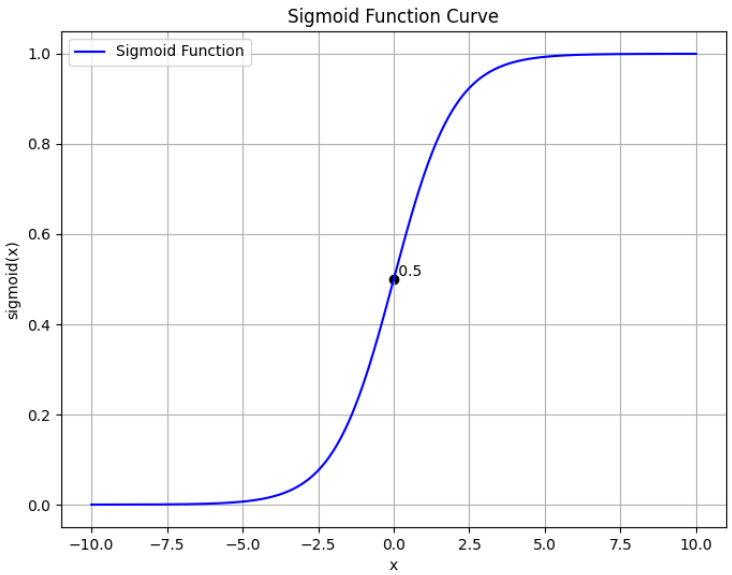


Figure 2.2: Sigmoid function curve.

Sigmoid Function

The sigmoid function is a type of activation function that is traditionally very popular. It is an S-shaped function that maps any input value into a range between 0 and 1. It is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.3}$$

It is often used in the output layer of a binary classification neural network. However, the effectiveness of the sigmoid function in deep networks is limited due to its susceptibility to the vanishing gradient problem, a situation where the gradients become extremely small during the backpropagation process.

Hyperbolic Tangent (tanh) Function

The tanh function is similar to the sigmoid function but maps any input value into a range between -1 and 1, providing a zero-centred output that helps mitigate biases in gradient

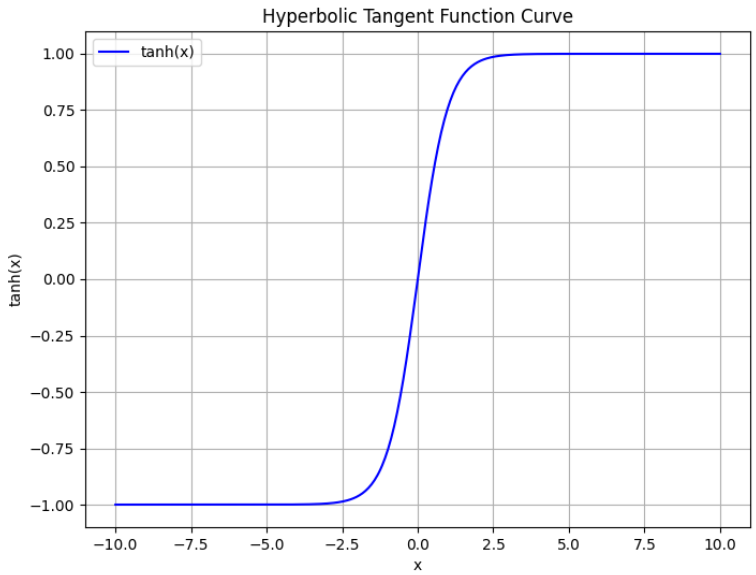


Figure 2.3: Hyperbolic tangent function curve.

updates during training. It is defined as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.4}$$

The property of centring around zero can lead to faster convergence compared to the sigmoid activation function, which outputs values between 0 and 1. However, the tanh function still has the same problem as the sigmoid function, which is susceptible to the vanishing gradient problem, particularly in deep networks. It is often used in the hidden layers of a neural network.

Rectified Linear Unit (ReLU) Function

The ReLU function has become one of the most widely used activation functions due to its simplicity and efficiency. It is defined as:

$$f(x) = max(0, x). \tag{2.5}$$

It outputs the input value directly if it is positive and zero otherwise. ReLU introduces non-linearity while maintaining computational efficiency. This characteristic allows ReLU to

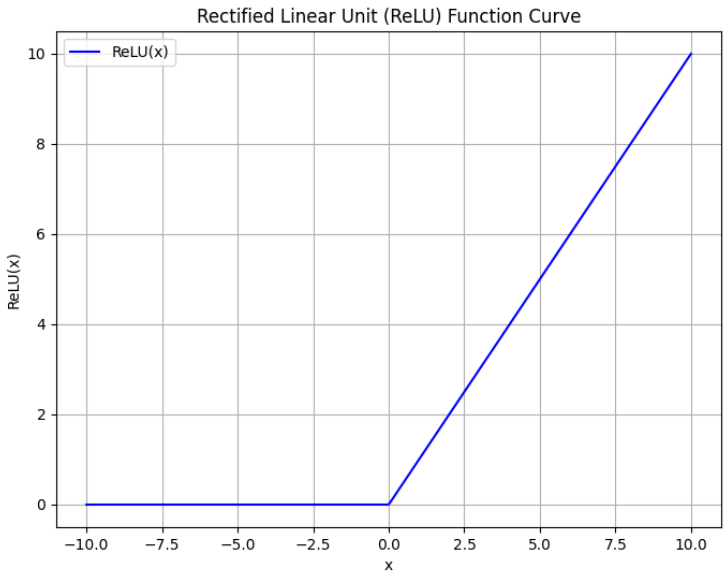


Figure 2.4: Rectified linear unit function curve.

address issues associated with vanishing gradients, a problem that can hinder the training of deep networks. However, ReLU has its drawbacks, such as the "dying ReLU" problem, where neurons can become inactive and stop learning if their inputs are predominantly negative. Despite this, its advantages have made ReLU a default choice in many modern neural network architectures.

Leaky Rectified Linear Unit Function

Leaky ReLU (Rectified Linear Unit) is a variant of the standard ReLU (Rectified Linear Unit) function. It addresses the "dying ReLU" problem of the traditional ReLU function by allowing a small, non-zero gradient when the input is negative. The Leaky ReLU function is defined as:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0. \end{cases} \tag{2.6}$$

Here, α is a small constant, usually $\alpha = 0.01$. This allows the function to have a small, non-zero output for negative input values, which helps keep the neurons active and mitigates the dying ReLU problem.

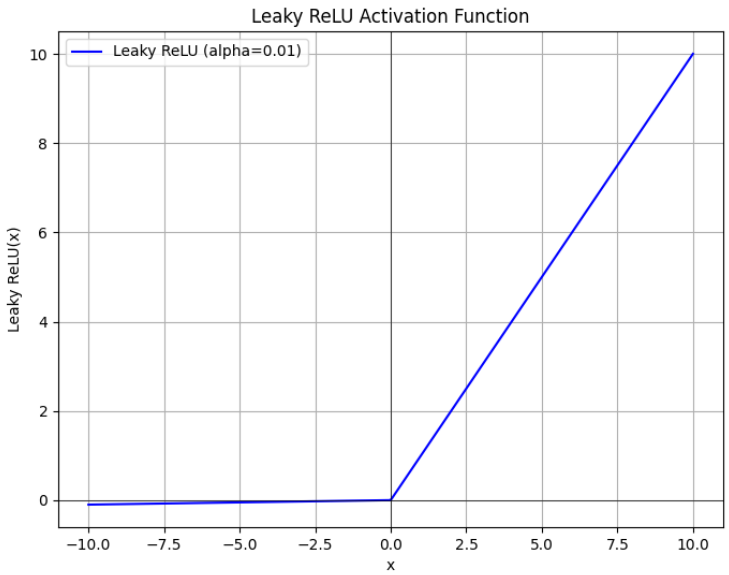


Figure 2.5: Leaky rectified linear unit function curve.

Softmax Function

The softmax function is a more generalized logistic activation function which is used for multiclass classification. It is defined as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \tag{2.7}$$

it transforms a vector of raw scores or logits into a probability distribution where each value lies between 0 and 1, and all the values sum to 1. This normalization ensures that the outputs can be interpreted as probabilities for each class, making it ideal for tasks where the network needs to assign an input to one of several categories. By emphasizing the relative differences between the input values, the softmax function helps in making clear and interpretable predictions in classification tasks.

2.2 Deep Learning Architectures

There are many deep learning architectures, each designed to address specific types of problems and data structures. These architectures range from traditional feedforward neural networks to more advanced and specialized models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), generative adversarial networks (GANs), and transformers. CNNs are particularly effective for image and spatial data processing, leveraging their ability to capture hierarchical patterns through convolutional layers. RNNs, on the other hand, excel in sequential data tasks by maintaining internal states that capture temporal dependencies. GANs consist of two neural networks, a generator and a discriminator, that are trained simultaneously through adversarial processes. The generator creates synthetic data samples, while the discriminator attempts to distinguish between real and generated samples, leading to high-quality data generation. Transformers, a more recent innovation, have revolutionized natural language processing with their self-attention mechanisms, enabling models to handle long-range dependencies and large-scale data efficiently. The development of these architectures has significantly advanced the capabilities of deep learning systems, allowing them to tackle increasingly complex and diverse challenges across various domains.

2.2.1 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a specialized deep learning architecture primarily used for image and video analysis. CNNs employ convolutional layers that apply filters to the input data, detecting local patterns such as edges, textures, and shapes. CNNs have achieved remarkable success in tasks like image classification, object detection, and facial recognition. There are several key components in CNNs, including convolutional layers, strides, padding, and pooling layers. In our model, CNN serves as the fundamental building block, leveraging its strength in capturing spatial features to enhance the quality of video reconstruction and analysis.

Convolutional Layer

The convolutional layer is the core building block of a CNN, responsible for detecting features in the input data. It applies a set of learnable filters (kernels) to the input image or feature map. The kernel size is commonly set to 1×1, 3×3 or 7×7. Each filter performs a convolution operation, where it slides over the input, computing dot products

between the filter values and the local patch of the input it covers, as illustrated in Fig. 2.6. For a given input image I and a filter K, the convolution operation to produce the output feature map O is given by:

$$O(i,j) = (I * K)(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) \cdot K(m,n), \tag{2.8}$$

where,

- I is the input image.
- K is the filter (or kernel).
- O is the output feature map.
- i and j are the spatial coordinates of the output.
- M and N are the dimensions of the filter.

This operation produces a feature map that highlights specific patterns such as edges or textures.

Stride

Stride refers to the number of pixels by which the filter moves across the input image or feature map during convolution. A stride of 1 means the filter moves one pixel at a time, ensuring detailed coverage of the input. Increasing the stride reduces the spatial dimensions of the output feature map, resulting in a more compact representation and reduced computational load. Specifically, if the stride of a filter is s, the filter moves s pixels at a time. The formula for the output dimensions with stride is:

$$O_{width} = \left\lfloor \frac{I_{width} - K_{width}}{s} \right\rfloor + 1, \tag{2.9}$$

$$O_{height} = \left\lfloor \frac{I_{height} - K_{height}}{s} \right\rfloor + 1. \tag{2.10}$$

Stride thus balances between feature resolution and computational efficiency.

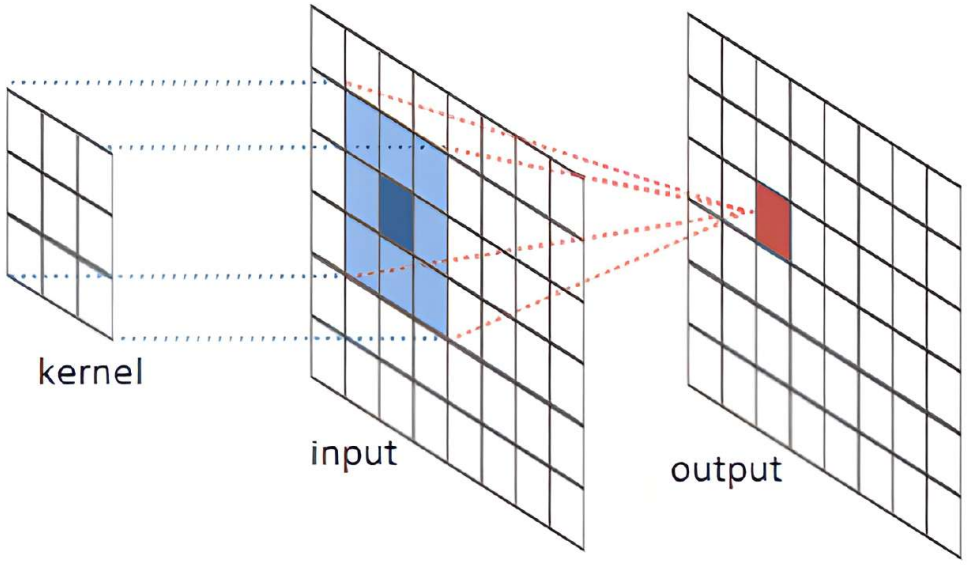


Figure 2.6: Convolutional layer operation, from [1].

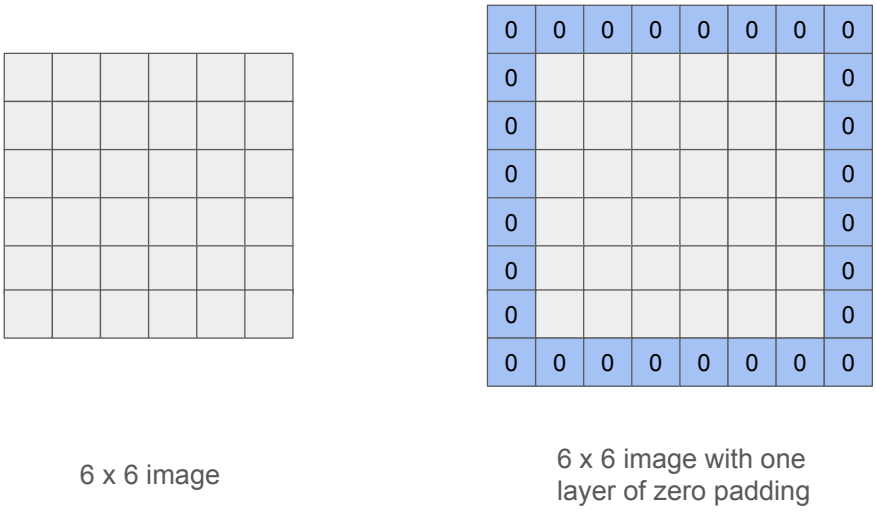


Figure 2.7: Padding.

Padding

Padding involves adding extra pixels around the border of the input image or feature map before applying the convolutional filter, as illustrated in Fig. 2.7. This is done to control the spatial dimensions of the output feature map and to ensure that features at the edges are not lost. Specifically, if padding p is applied, the input image is padded with zeros around the border. The formula for the output dimensions with padding is:

$$O_{width} = \left\lfloor \frac{I_{width} - K_{width} + 2p}{s} \right\rfloor + 1, \tag{2.11}$$

$$O_{height} = \left\lfloor \frac{I_{height} - K_{height} + 2p}{s} \right\rfloor + 1. \tag{2.12}$$

Pooling layer

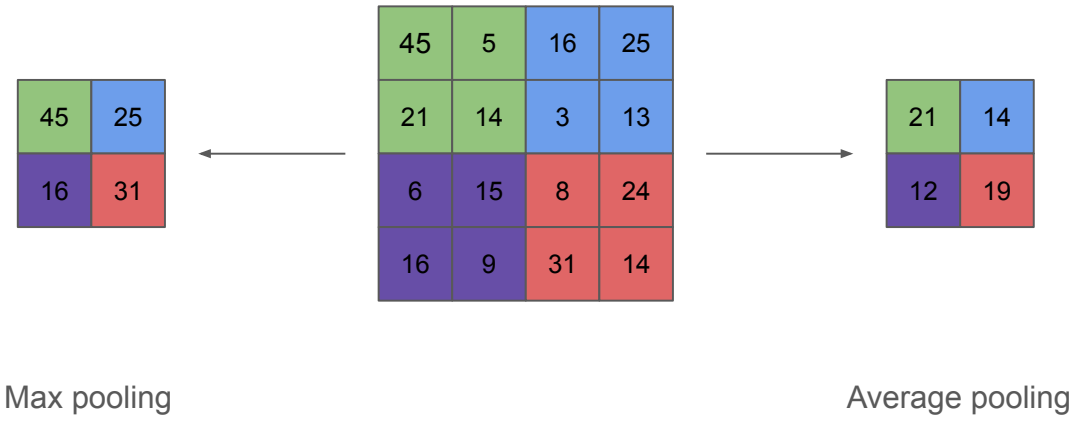


Figure 2.8: Pooling.

The pooling layer in Convolutional Neural Networks (CNNs) is designed to downsample feature maps, reducing their spatial dimensions while retaining the most essential information. Typically using operations like max pooling or average pooling, this layer selects the maximum or average value from a specified window, 2×2 or 3×3, within the feature map.

Specific operations of max pooling and average pooling are shown in Fig. 2.8. By doing so, pooling layers not only decrease the computational load and memory usage but also help in making the network more robust to small translations and distortions in the input data. This reduction in dimensionality also helps in controlling overfitting and extracting high-level features, facilitating more efficient and effective training of the network.

2.2.2 Recurrent Neural Network (RNN)

Recurrent neural network (RNN) is a class of artificial neural networks designed to handle sequential data by incorporating temporal dynamics into their architecture. Unlike traditional convolutional neural networks, RNNs possess connections that form directed cycles, allowing them to maintain a state or memory of previous inputs, as shown in Fig. 2.9. This temporal memory makes RNNs particularly well-suited for tasks involving sequences, such as natural language processing, speech recognition, and time series forecasting. In our model, RNN plays a critical role, working in conjunction with CNN to effectively capture both spatial and temporal features, enabling robust video reconstruction and analysis.

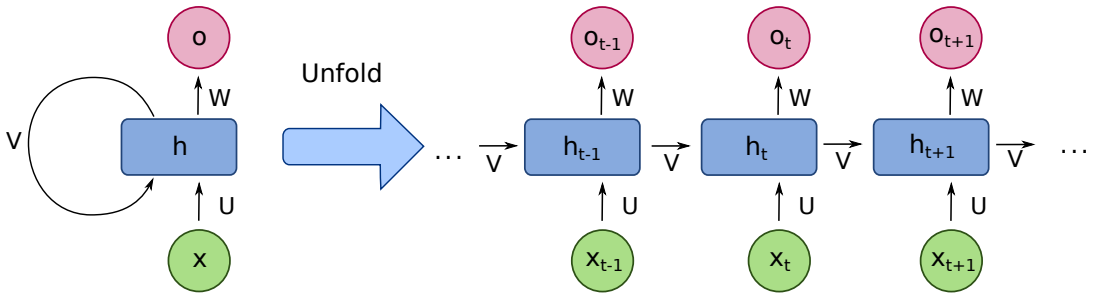


Figure 2.9: Recurrent neural network, from [2].

At the core of an RNN is the concept of recurrence, where the network's hidden state is updated iteratively as it processes each element in the sequence. This hidden state acts as a form of memory, capturing information from past inputs and influencing the network's responses to future inputs. Mathematically, this is represented as:

$$h_t = f(Vh_{t-1} + Ux_t + b_h), \tag{2.13}$$

where,

- h_t is the hidden state at time step t.
- h_{t-1} is the hidden state from the previous time step.
- x_t is the input at time step t.
- V is the weight matrix transforms the previous hidden state.
- U is the weight matrice transforms the current input.
- b_h is the bias term.
- f is an activation function, such as tanh or ReLU.

Despite their effectiveness, standard RNNs struggle with long-term dependencies due to issues like vanishing and exploding gradients. To address these challenges, advanced RNN architectures such as long short-term memory (LSTM) networks and gated recurrent units (GRUs) have been developed. These variants include mechanisms to better capture and retain long-term dependencies, significantly improving performance in tasks requiring the modelling of long sequences.

Overall, RNNs and their advanced forms play a crucial role in modern machine learning by enabling models to understand and generate sequences of data, making them indispensable in various applications that involve time-dependent information.

2.2.3 Generative Adversarial Network (GAN)

Generative adversarial network (GAN) is a class of machine learning frameworks introduced by Goodfellow *et al.* [25], designed to generate synthetic data that is indistinguishable from real data. GANs consist of two neural networks, a generator and a discriminator, that compete against each other in a game-theoretic framework, as shown in Fig. 2.10. The generator aims to create realistic data samples, while the discriminator attempts to distinguish between real and generated samples. Through this adversarial process, the generator progressively improves its ability to produce convincing data, such as images, music, or text. GANs have gained significant attention for their ability to generate high-quality synthetic data, with applications in image synthesis, style transfer, and data augmentation.

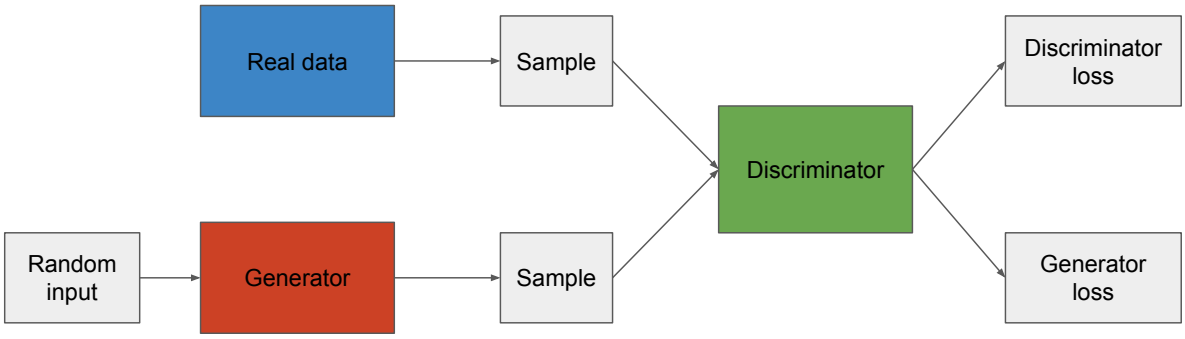


Figure 2.10: Generative Adversarial Networks (GANs).

2.2.4 Transformers

In the field of deep learning, transformers have revolutionized natural language processing (NLP) and beyond, becoming one of the most influential architectures in recent years. Introduced by Vaswani *et al.* in their paper, "Attention is All You Need" [26], transformers have transformed how machine learning models understand and generate sequences of data. Unlike traditional sequence models like RNN and LSTM, transformers do not rely on sequential processing, making them more efficient and scalable for handling large datasets.

Self-Attention Mechanism

The self-attention mechanism is a fundamental component of the transformer architecture, enabling the model to weigh the importance of each token in a sequence relative to others. It begins by projecting the input matrix X into three distinct vectors—Query Q, Key K, and Value V—using learned weight matrices W^Q, W^K, W^V, respectively, as follows:

$$Q = XW^Q, \tag{2.14}$$

$$K = XW^K, \tag{2.15}$$

$$V = XW^V. \tag{2.16}$$

Attention scores are computed by taking the dot product of Query vectors with Key vectors and scaling by the square root of the Key dimension d_k, given by

$$\text{Attention\_Score}_{ij} = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}}, \tag{2.17}$$

These scores are normalized using the softmax function to produce attention weights:

$$\text{Attention\_Weights}_{ij} = \text{softmax}(\text{Attention\_Score}_{ij}). \tag{2.18}$$

Finally, the output for each token is obtained as a weighted sum of the Value vectors, with the attention weights serving as coefficients:

$$\text{Output}_i = \sum_j \text{Attention\_Weights}_{ij} \cdot V_j. \tag{2.19}$$

This process allows each token to dynamically attend to and integrate information from all other tokens in the sequence.

Positional Encoding

Since transformers do not have an inherent notion of sequential order, positional encodings are added to the input embeddings to provide information about the position of each token in the sequence. These encodings enable the model to understand the order of tokens, which is crucial for tasks like translation or text generation.

Multi-Head Attention

To capture different aspects of the relationships between tokens, transformers use multi-head attention. This mechanism involves running multiple self-attention operations in parallel, allowing the model to focus on different parts of the sequence simultaneously and learn diverse features.

Encoder-Decoder Architecture

The original transformer model comprises an encoder-decoder structure. The encoder processes the input sequence and generates a set of representations, while the decoder uses these representations to produce the output sequence. This design is particularly effective for tasks like machine translation, where there is a clear input-output relationship.

2.3 Training Neural Networks

Training neural networks involves optimizing a model's parameters to minimize the error between its predictions and actual outcomes using large datasets. This process, known as backpropagation, involves feeding data through the model to adjust its parameters iteratively, enabling it to make increasingly accurate predictions. During training, the neural network learns complex patterns and features from the input data through multiple layers of neurons, with each layer progressively capturing higher-level abstractions. Effective training is crucial for developing models that generalize well to new, unseen data, thereby enabling accurate and reliable predictions in various applications. In the following sections, we will introduce several key components of the training process, including data preparation, loss functions, optimization algorithms, and hyperparameters.

2.3.1 Dataset Preparation

Data preparation is a crucial step in training neural networks, as the quality and structure of the input data significantly impact the model's performance. The process begins with data collection, where a large and representative dataset is gathered. This dataset often requires cleaning to remove noise and inconsistencies, such as missing values, duplicates, and outliers. Cleaning ensures that the data fed into the neural network is accurate and reliable.

Next, the data is typically normalized or standardized to bring all features to a similar scale, which helps improve the efficiency and effectiveness of the training process. For instance, features with different ranges can lead to unstable gradients during backpropagation, making it harder for the model to converge. Additionally, data may be augmented or transformed to enhance diversity and robustness. Techniques such as rotation, scaling, and flipping are common in image data augmentation, while text data might undergo tokenization and stemming. Finally, the prepared data is often split into training, validation,

and test sets to evaluate the model's performance and generalization ability accurately. Proper data preparation ensures that the neural network can learn meaningful patterns and perform well on unseen data.

2.3.2 Loss Functions

Loss function is a mathematical method used to estimate the discrepancies between the predicted and actual outcomes. It quantifies the error of the prediction, providing a measure that the learning algorithm seeks to minimize during training. Different types of loss functions are used depending on the nature of the task, such as Mean Squared Error for regression tasks, or Cross-Entropy for classification tasks. The optimization of the loss function using techniques like gradient descent is central to the training of deep learning models. We will introduce several common loss functions.

Mean Squared Error (MSE)

The Mean squared error is used primarily for regression tasks. It measures the average squared difference between predicted values and actual values. Its formula is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.20}$$

where,

- n is the number of observations.
- y_i is the actual value for the i-th observation.
- $\hat{y}_i$ is the predicted value for the i-th observation.

Mean Absolute Error (MAE)

The Mean absolute error is another metric for regression tasks. It measures the average magnitude of errors in predictions, without considering their direction. Its formula is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{2.21}$$

Cross-Entropy

Cross-entropy is used for classification tasks where the goal is to classify inputs into one of several categories. It measures the performance of a classification model whose output is a probability value between 0 and 1.

$$\text{Categorical\_Cross\_Entropy} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}), \tag{2.22}$$

where,

- y_i is the binary indicator (0 or 1) if class label.
- i is the correct classification.
- $\hat{y}_i$ is the predicted probability of class i.

2.3.3 Optimization Algorithms

Optimization algorithms are crucial in training neural networks as they minimize the loss function and adjust the model's weights to improve performance.

Gradient Descent (GD)

Gradient descent (GD) is a foundational optimization algorithm that updates the model's parameters iteratively by moving them in the direction of the negative gradient of the loss function. The formula for Gradient Descent is:

$$\theta_{t+1} = \theta_t - \alpha\nabla_\theta J(\theta_t), \tag{2.23}$$

where,

- θ_t represents the model's parameters at iteration t.
- α is the learning rate.
- $\nabla_\theta J(\theta_t)$ is the gradient of the loss function with respect to θ at iteration t.

It is simple and intuitive to understand and implement, making it a foundational technique in machine learning. However, it can be slow, especially for large datasets, and is prone to getting stuck in local minima.

Stochastic Gradient Descent (SGD)

Stochastic gradient descent (SGD) is a variant of GD where the model's parameters are updated for each training example rather than the entire dataset. Its formula is:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta_t; x^{(i)}, y^{(i)}), \tag{2.24}$$

where $x^{(i)}$ and $y^{(i)}$ are the i-th training example and its corresponding label. This can lead to faster convergence, particularly for large datasets. The noisy updates of SGD can help escape local minima, and it requires less memory since it processes one example at a time. However, the noise in the updates can cause the loss function to fluctuate, making SGD less stable compared to gradient descent.

Momentum

Momentum is used to accelerate SGD in the relevant direction and dampen oscillations by accumulating a velocity vector in directions of persistent reduction in the loss function. The formulas for Momentum are:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla_\theta J(\theta_t), \tag{2.25}$$

$$\theta_{t+1} = \theta_t - \alpha v_{t+1}, \tag{2.26}$$

where v_t is the velocity update and β is the momentum parameter. The main advantage of momentum is its ability to smooth out the optimization path and speed up convergence. However, it introduces an additional hyperparameter β that requires tuning. If the momentum term is too large, it may cause the optimization process to overshoot, leading to instability.

RMSProp

RMSProp [27] is an adaptive learning rate method designed to perform well in non-stationary environments by maintaining a moving average of the squared gradients. The formulas for RMSProp are:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)(\nabla_\theta J(\theta_t))^2, \tag{2.27}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta J(\theta_t). \tag{2.28}$$

Adam (Adaptive Moment Estimation)

Adam [28], which stands for Adaptive Moment Estimation, is an adaptive learning rate algorithm designed to improve training speeds in deep neural networks and reach convergence quickly. It computes individual adaptive learning rates for different parameters. The formulas for Adam are:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta_{t-1}), \tag{2.29}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta_{t-1}))^2, \tag{2.30}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.31}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2.32}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{2.33}$$

where β_1 and β_2 are hyperparameters typically set to 0.9 and 0.999. Adam's combination of adaptive learning rates, momentum, and bias correction makes it a highly effective and versatile optimization algorithm, contributing significantly to its widespread adoption in various deep-learning tasks.

2.3.4 Hyperparameters

Hyperparameters are crucial elements in training neural networks, influencing the network's performance and learning efficiency. Unlike model parameters, which are learned from the data (such as weights and biases), hyperparameters are set before the training process begins. They control various aspects of the learning process and the network's architecture. Here are several common hyperparameters:

- **Learning rate:** It controls the size of the steps taken towards the minimum of the loss function during training. A smaller learning rate may lead to slower convergence, while a larger learning rate can cause overshooting.

- **Batch size:** The number of training examples used in one iteration to update the model's parameters. Larger batch sizes can lead to more stable gradients but require more memory.
- **Number of epochs:** The number of times the entire training dataset is passed through the model during training. More epochs allow the model to learn more but may lead to overfitting.
- **Initialization method:** It refers to the strategy used to set the initial values of the network's weights before the learning process begins. It can influence the convergence speed and overall performance of the network. There are some popular initialization methods, such as random initialization, Xavier initialization [29] and He initialization [30].

Finding the right hyperparameters involves a combination of experimentation, intuition, and systematic exploration. Start by selecting a range of plausible values for each hyperparameter based on domain knowledge and previous experience. Use techniques such as grid search or random search to evaluate different combinations of these values, leveraging cross-validation to assess performance on validation data. For more efficient exploration, consider advanced methods like Bayesian optimization, which models the performance as a probabilistic function of hyperparameters and intelligently selects the next set of parameters to test. Continuously monitor the model's performance and adjust the search strategy based on the results to fine-tune the hyperparameters for optimal results.

Chapter 3

Literature Review

3.1 Traditional Methods

Traditional deinterlacing methods use straightforward techniques like blending or interpolating fields based on spatial and temporal patterns. While these methods are computationally efficient and have been widely used, they often struggle with complex motion or detailed textures, leading to artifacts like blurring, aliasing, and loss of details. Traditional deinterlacing methods can generally be divided into three categories: edge-directional algorithms [31–36], motion-adaptive algorithms [4, 37–42], and motion-compensated algorithms [5, 43–49]. Edge-directional algorithms are fast but can struggle with complex motion; motion-adaptive algorithms represent a middle ground between speed and quality; and motion-compensated algorithms offer the highest quality by accurately tracking motions but require more processing time and computational resources.

3.1.1 Edge-directional Algorithms

Edge-directional algorithms [31–36] focus on the identification and analysis of edges within frames. As shown in Fig. 3.1, the algorithm begins by detecting edges within the frame. Edges are areas where there is a significant contrast between neighbouring pixels, and they are crucial for preserving detail. Once edges are detected, the algorithm analyzes their direction. Understanding the direction of these edges is essential because it informs how the frame should be reconstructed, ensuring that details are maintained and artifacts are minimized. Following edge detection and directional analysis, the algorithm performs

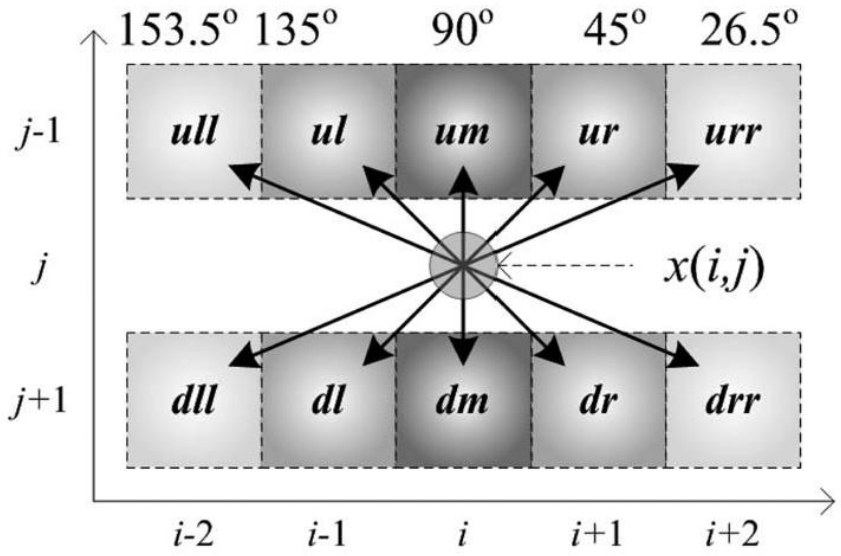


Figure 3.1: Edge-directional algorithm, from [3].

edge-aligned interpolation. This means that pixel adjustments are made in a way that aligns with the detected edges, helping to preserve the integrity of the frame details. By combining the two fields from the interlaced frame into a single progressive frame with this edge-directional approach, the result is a clearer and more visually appealing frame compared to simpler deinterlacing methods.

To delve into the specifics of edge-directional algorithms, let's explore some contributions to this domain. Michaud *et al.* [31] introduced a fuzzy edge-direction detector designed to improve the performance of video line doubling, particularly for moving diagonal lines. The detector identifies small pixel variations in five orientations (0°, ±45°, and ±60°) and uses fuzzy logic rules to infer the prevailing edge direction. Park *et al.* [33] presented an algorithm focused on detecting and preserving horizontal edge patterns to improve the visual quality of deinterlaced video. It uses edge-dependent interpolation to accurately estimate and reconstruct missing pixels along horizontal edges, reducing artifacts and enhancing edge clarity. Part *et al.* [35] proposed a method to improve the quality of deinterlaced images by analyzing the distribution of Discrete Cosine Transform (DCT) coefficients. The algorithm uses edge direction information derived from these coefficients to guide the interpolation process. Yang *et al.* [36] proposed an algorithm that employs

horizontal and vertical gradient operators to detect various edges and determines the dominant edge direction by analyzing the surrounding edges of the interpolating pixel. Yoo *et al.* [32] proposed a method that introduces the upper spatial direction vector (USDV) and lower spatial direction vector (LSDV) to better determine the direction of the highest spatial correlation. By identifying these edge directions, the algorithm can more accurately interpolate the missing lines, preserving the details and reducing artifacts. Jin *et al.* [34] presented a deinterlacing algorithm using a modified Sobel operation, which is designed to enhance the quality of deinterlacing by accurately detecting and preserving edges in video frames. The process begins with edge detection, where the algorithm identifies edge regions in the video frame using a modified Sobel operation. Once the edges are detected, the algorithm selects Candidate Direction Vectors (CDVs) based on the detected edges. These CDVs represent the possible directions of the edges, providing a framework for the next phase. The fine directional interpolation is then performed using the selected CDVs, which aids in accurately reconstructing the missing lines in the interlaced video frames. This method is particularly effective in preserving details and reducing artifacts in the deinterlaced video, making it superior to traditional edge-directional deinterlacing algorithms.

3.1.2 Motion-adaptive Algorithms

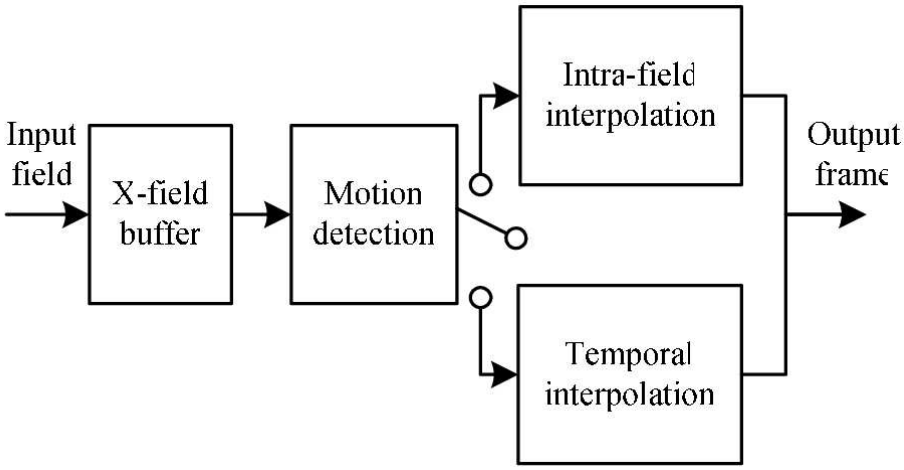


Figure 3.2: Motion-adaptive algorithm, from [4].

Motion-adaptive algorithms [4,37–42] selectively apply spatial or temporal interpolation

to different regions of an interlaced video frame based on the amount of motion detected in that region.

As shown in Fig. 3.2, the first step in motion-adaptive deinterlacing involves detecting motion between the fields of the video. This is achieved by comparing corresponding areas in the odd and even fields to identify significant movement. Once motion is detected, the algorithm applies different processing methods based on the presence of motion. In areas with little or no motion, simpler methods such as field duplication, where one field is used to fill in the missing lines of the other, are used. However, in areas with significant motion, more sophisticated techniques are employed to preserve frame quality and reduce artifacts. These techniques may include motion-compensated interpolation, which estimates pixel positions in the new frame based on the detected movement, and edge-directed interpolation, which adapts the interpolation process to preserve edges and details in the video.

Having outlined the fundamental principles of motion-adaptive algorithms, it is important to delve into specific studies that have advanced this field. Brox *et al.* [41] presented an algorithm that employs a simple fuzzy inference system to handle picture repetition, ensuring smooth transitions and reducing artifacts. Choi *et al.* [42] proposed a method that employed different neural networks based on the amount of motion detected in the video frames1. By selectively using these networks, the algorithm can effectively handle variations in motion between adjacent fields. Wang *et al.* [38] presented a method that combines two existing deinterlacing techniques: one that provides high vertical resolution and another that is robust to erroneous motion vectors. The method uses a new measurement of motion vector reliability, switching between the techniques based on this reliability. Shen *et al.* [39] proposed a technique that combines spatial and temporal filtering to improve picture quality, using advanced motion detection to differentiate between static and dynamic regions. Yu *et al.* [4] presented a new accurate motion detection algorithm to improve the accuracy of motion detection in deinterlacing, which reduces errors using a median filter. Lin *et al.* [37] introduced a motion adaptive method with horizontal motion detection. The algorithm has four key features:

- ELA-Median Directional Interpolation: This technique enhances the sharpness of edges in the image by using edge-based line averaging (ELA) combined with median processing. This helps in producing clearer and more defined images.
- Same-Parity 4-Field Horizontal Motion Detection: This method detects faster motion by analyzing four fields with the same parity. It improves the accuracy of motion detection, which is crucial for determining the correct interpolation method to use.

- Morphological Operation for Noise Reduction: This feature involves using morphological operations to reduce noise in the image. It helps in preserving the actual texture of the original objects, leading to higher quality deinterlaced images.
- Adaptive Threshold Adjusting: This technique dynamically adjusts the threshold levels based on the content of the image. It ensures that the deinterlacing process adapts to different types of motion and image characteristics, resulting in a better overall image quality.

3.1.3 Motion-compensated Algorithms

Motion-compensated deinterlacing improves upon traditional methods by incorporating motion estimation and compensation techniques. Initially, the algorithm analyzes the video to detect motion between adjacent fields or frames. This involves comparing the current field with past and future fields to estimate how pixels have shifted. The calculated motion vectors represent the direction and distance of pixel movement, which is crucial for accurate reconstruction. Once motion is estimated, the algorithm uses this information to predict the appearance of missing lines in the current field. By integrating these predictions, the algorithm creates a full, progressive frame. This method effectively reduces common artifacts like combing and blurring, which are often seen with simpler deinterlacing techniques, resulting in a smoother and more accurate video representation.

Motion-compensated algorithms [5,43–49] estimate motion to interpolate missing fields in interlaced video, resulting in smoother and more accurate frames. Li *et al.* [43] introduced a method that uses a phase-correction filter applied to either even or odd fields before motion detection. Chang *et al.* [44] presented an algorithm that uses block-based directional edge interpolation and same-parity 4-field motion detection to improve image quality. Mohammadi *et al.* [46] proposed a novel deinterlacing method that uses bi-directional motion estimation, leveraging two previous and two subsequent fields. Li *et al.* [47] formulated the deinterlacing process as a maximum a posteriori problem and addressed it using a Markov random field model. Chen *et al.* [48] presented an algorithm that employs forward, backward, and bi-directional motion estimation with sub-pixel accuracy to improve the quality of the deinterlaced video. Huang *et al.* [5] presented a motion-compensated deinterlacing scheme, as shown in Fig. 3.3, that leverages hierarchical motion analysis to enhance the quality of video conversion from interlaced to progressive formats. The core idea is to detect and utilize motion information at various levels of granularity, starting with large partitions and refining down to smaller sub-blocks. This hierarchical approach

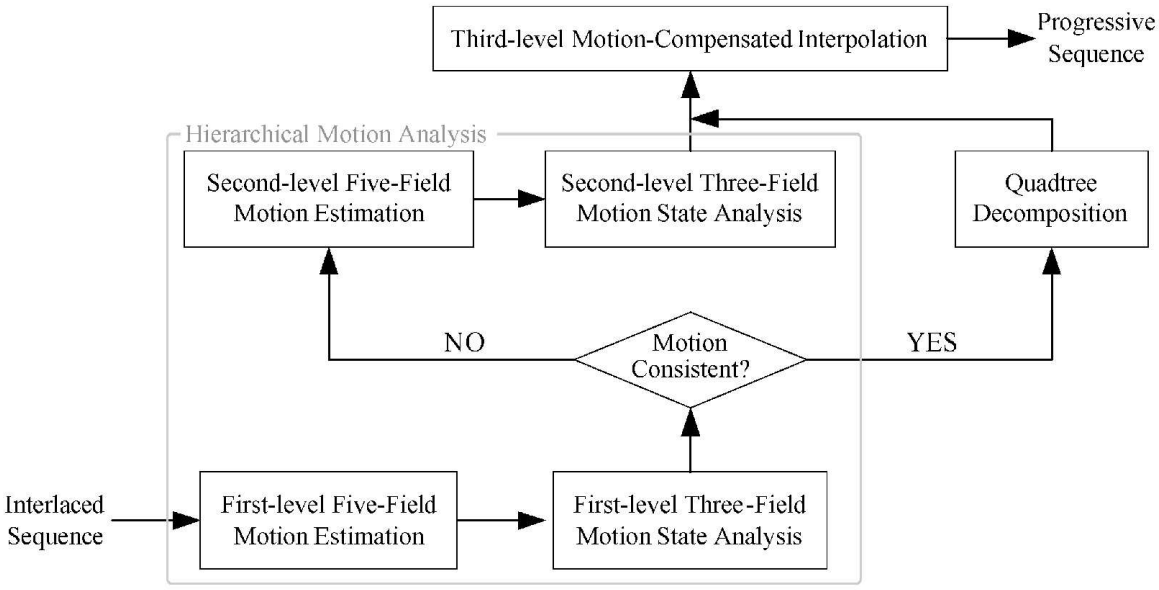


Figure 3.3: Motion-compensated algorithm, from [5].

allows for accurate motion detection across a wide range of motion activities, from slow to fast movements.

The advantages of motion-compensated deinterlacing are notable. It minimizes artifacts and enhances frame quality, particularly for dynamic scenes with fast movement. This makes it particularly beneficial for sports broadcasts and action films where motion accuracy is critical. However, the process is computationally demanding, requiring significant processing power and memory, which can be a constraint in certain hardware environments. Additionally, in highly complex scenes or noisy footage, some artifacts might still occur.

3.2 Deep Learning Based Methods

Compared to traditional methods, deep learning based approaches leverage the power of neural networks to learn and predict how to reconstruct missing information from interlaced frames. By training on large datasets of high-quality video, these models can capture intricate patterns and relationships in the data that traditional methods may miss.

3.2.1 Intra-frame methods

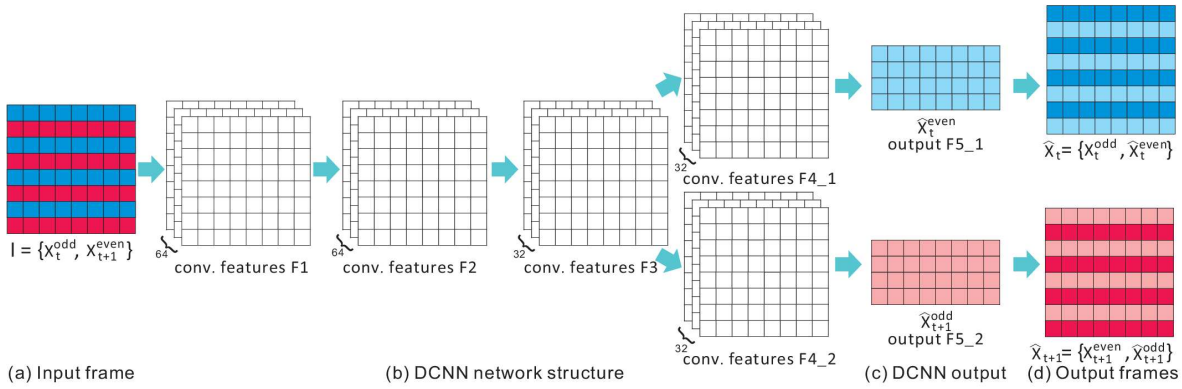


Figure 3.4: DICNN, from [6].

Intra-frame methods [6], [7], [8] in deinterlacing focus on processing each frame individually to convert interlaced video into a progressive format. These methods work by analyzing and reconstructing the frame within a single frame, rather than relying on information from adjacent frames. An advantage of intra-frame methods is their simplicity and parallelizability. Since each frame is processed independently, these methods can be implemented efficiently and scaled easily across multiple frames. However, this approach ignores the temporal context provided by adjacent frames. This can lead to artifacts or inconsistencies at the boundaries of deinterlaced frames, as there is no mechanism to ensure that motion or changes between frames are smooth and coherent. Especially in scenes with significant motion, intra-frame methods might fail to reconstruct details that are affected by movement, resulting in distorted deinterlacing results.

Zhu *et al.* [6] presented the first deep convolutional neural network based method (DICNN) for deinterlacing, which has five convolutional layers as shown in Fig. 3.4. In the figure, the network extracts information from both odd and even fields. In the fourth layer the network branches into two distinct paths to generate two missing fields. Finally, these generated fields are combined with the input fields to produce two output frames. Experimental results demonstrate that this deep learning-based method outperforms traditional deinterlacing techniques in terms of both reconstruction accuracy and computational efficiency. However, the network structure is relatively simple, which may limit its ability to capture complex patterns in the data.

Liu *et al.* [7] further introduced deformable convolution and residual blocks to DICNN,

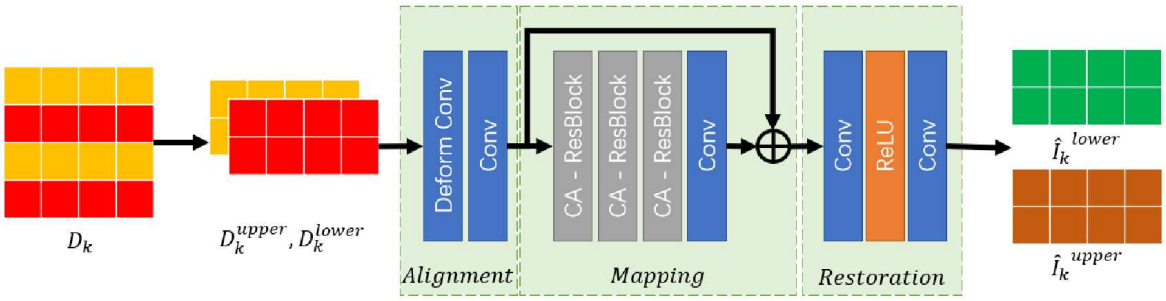


Figure 3.5: Liu's network structure, from [7].

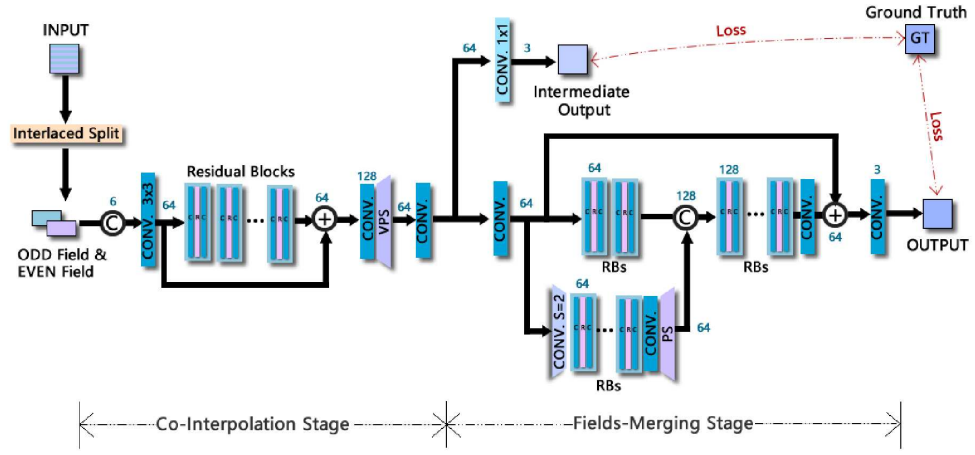


Figure 3.6: Zhao's network structure, from [8].

as shown in Fig. 3.5. Deformable convolutions in the alignment module are used to enhance the model's ability to align features across two fields more accurately by allowing the convolutional kernels to adaptively adjust their spatial locations. This flexibility is crucial for handling motion and spatial misalignment, as it enables the model to capture and correct complex deformations and motion patterns that traditional convolutions might miss. Residual blocks introduce skip connections that bypass one or more layers, allowing the network to learn differences between the input and output of these blocks—rather than learning the full transformation directly. This architecture helps mitigate the vanishing gradient problem, facilitates deeper network training, and improves feature extraction by enabling the model to focus on details and subtle changes in the video data, ultimately leading to more effective restoration of high-quality video content.

Zhao *et al.* [8] proposed a deinterlacing network (DIN), which consists of two main stages: Co-Interpolation Stage and Fields-Merging Stage, as shown in Fig. 3.6. In the Co-Interpolation stage, the network focuses on splitting fields and utilizing adjacent field information to enhance the vertical resolution of the video. This stage aims to preserve the details and reduce the loss of information that typically occurs during deinterlacing. The Fields-Merging is designed to perceive movements and remove ghost artifacts that are common in early interlaced videos. By effectively identifying and addressing these artifacts, the network can produce cleaner and more visually appealing results.

3.2.2 Inter-frame methods

Inter-frame methods in video restoration focus on utilizing the temporal information from multiple frames within a video sequence to enhance or reconstruct each frame. These techniques analyze the relationship between consecutive frames to better reconstruct a progressive frame, aiming to provide a more accurate representation of motion and detail. The primary advantage of inter-frame methods is their ability to utilize temporal coherence — the fact that adjacent frames in a video are often similar or related. By integrating information across these frames, inter-frame methods can achieve a more comprehensive deinterlacing. This temporal context allows for better handling of motion artifacts and can provide more accurate restoration of details that span across frames, resulting in a smoother and more coherent visual output.

In the methods considering inter-frame information, Bernasconi *et al.* [23] applied the sliding-window framework for the alignment of neighbouring fields and used a residual dense network to generate a reconstructed frame.

VDNet [9] consists of two main components: the Data Module and the Residual, as

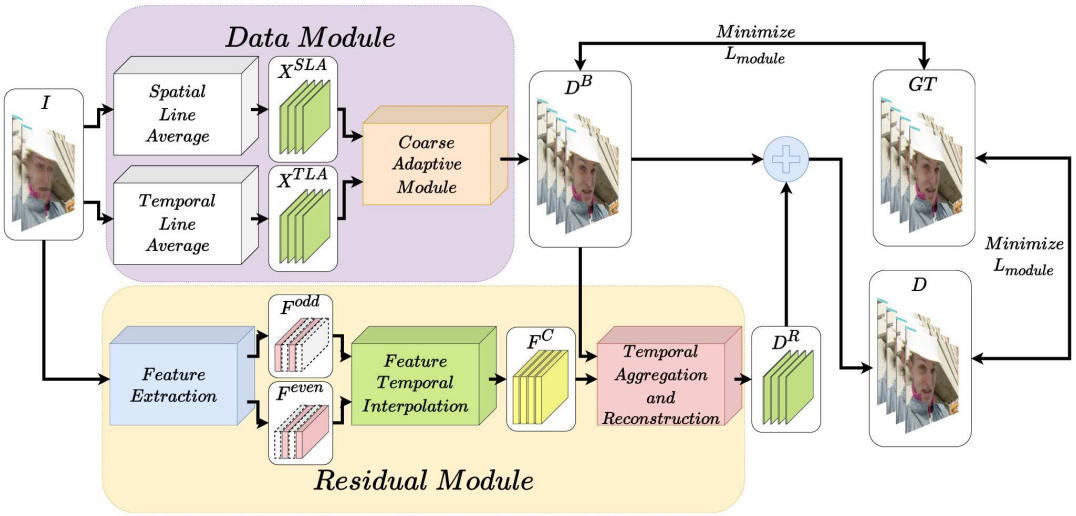


Figure 3.7: VDNet, from [9].

shown in Fig. 3.7. The Data Module initially processes the interlaced frames by splitting them and regenerating the missing lines using two traditional deinterlacing methods, spatial line average and temporal line average, and the Coarse Adaptive Module. This preliminary step helps in creating a base frame sequence. The Temporal Aggregation and Reconstruction Module uses a bidirectional deformable RNN to refine the base sequence

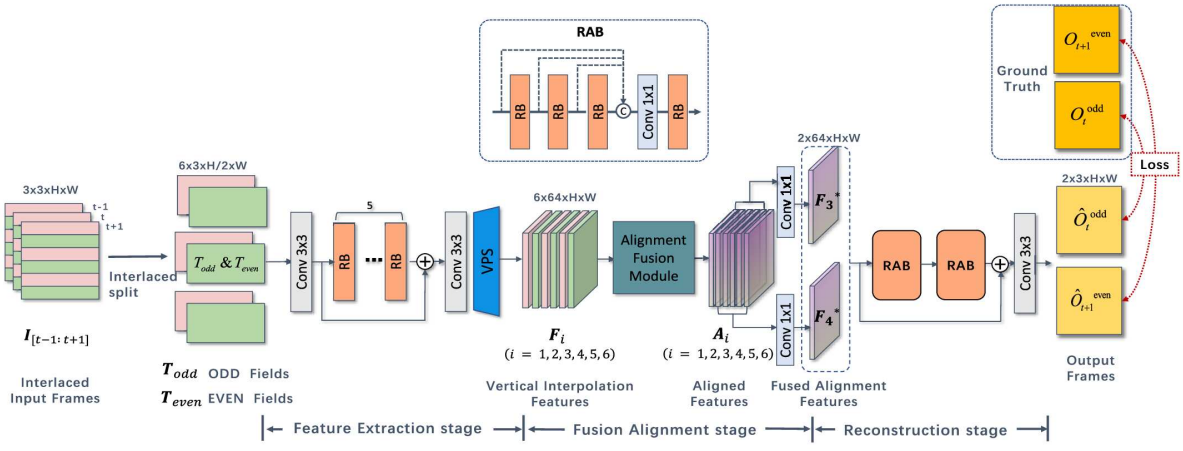


Figure 3.8: Multiframe deinterlacing network, from [10].

by enhancing and aggregating features extracted from the interlaced video. In addition, a key innovation in VDNet is the use of a Spatial-temporal correlation loss function. This loss function utilizes information from the existing interlaced video to smooth and enhance the deinterlaced output, effectively reducing artifacts and preserving details. The combination of these components allows VDNet to handle complex motion and deformation present in video sequences, ensuring a more accurate and detailed deinterlacing process. The disadvantage of the model is with so many different blocks, the model becomes overly complex.

Zhao [10] *et al.* propose a multi-frame deinterlacing network, which consists of three main stages: the Feature Extraction stage, the Fusion Alignment stage, and the Reconstruction stage, as shown in Fig. 3.8. The Feature Extraction stage is responsible for handling the vertical interpolation of interlaced frames, which is crucial for reconstructing high-quality frames from interlaced video data. The Fusion Alignment stage aligns and fuses temporal information from multiple frames, leveraging the redundancy in the temporal domain to enhance the video quality. The Fusion Alignment stage utilizes DCN-based implicit alignment, inspired by EDVR [17]. Finally, the Final Refinement Module refines the output to remove any remaining artifacts and improve the overall visual quality.

Gao *et al.* [11] proposed a method that leverages bidirectional spatio-temporal information propagation across multiple scales, as shown in Fig. 3.9, which helps to enhance the quality of legacy video content by effectively utilizing information in both space and time. The proposed method operates across multiple scales, allowing it to capture both fine details and broader contextual information. This multi-scale approach helps in accurately reconstructing high-quality frames from interlaced video. The authors demonstrate that their method outperforms existing deinterlacing techniques, both in terms of visual quality and quantitative metrics.

In all models considering inter-frame information above, there is a significant difference in the methodologies employed. They utilize different propagation methods - sliding-window [10], [23], bidirectional Deformable RNN [9] or bidirectional propagation in a UNet-like structure at different scales [11]. Distinct methodologies can yield varying results, potentially altering the video quality and execution time of the result.

3.2.3 Reconstruction

The reconstruction module is a critical and challenging component in deep learning based deinterlacing models. Because each frame in deinterlacing videos consists of two fields with alternating odd and even lines captured at different times, leading to spatial and temporal

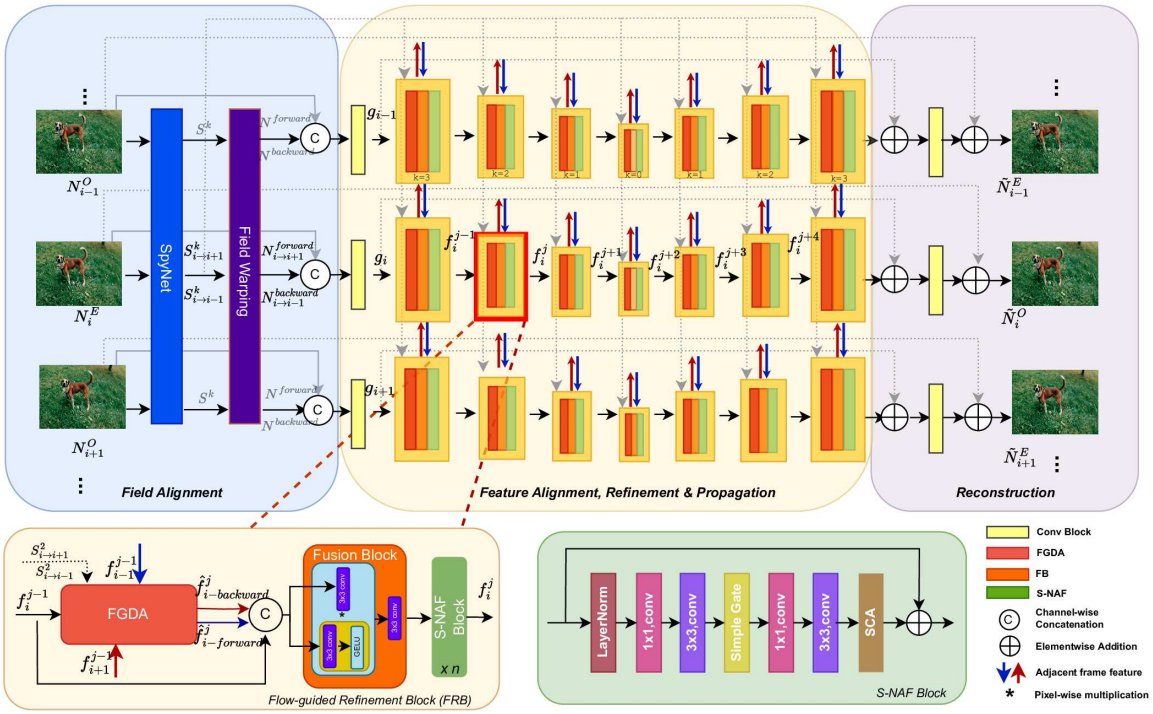


Figure 3.9: Gao's network structure, from [11]

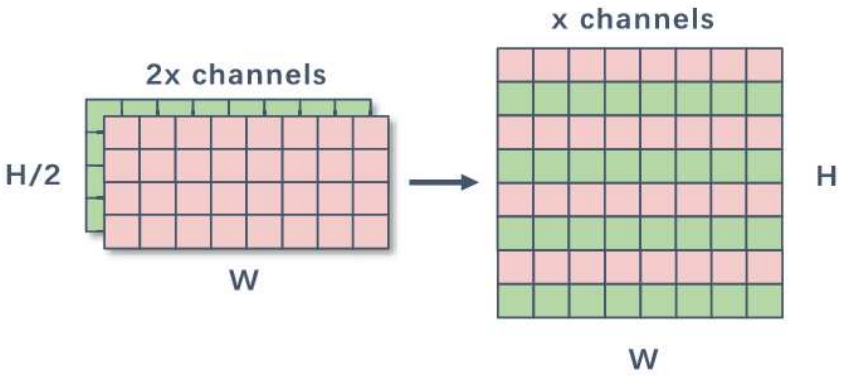


Figure 3.10: Vertical pixel shuffle, from [10].

discrepancies. The core of the reconstruction module is to synthesize full progressive frames from the processed features of fields. This involves generating a coherent and high-quality output frame that combines the best information from fields. The existing deinterlacing models typically employ three primary reconstruction methods.

The first method involves predicting the missing field directly, as demonstrated in works such as [6], [11], [7], [23]. Models take fields from interlaced video as input. The input sequence for the fields starts with odd fields in the first frame, then switches to even fields in the second frame, and continues alternating between odd and even fields for the following frames. Then the models output the missing half fields, starting from even fields for the first frame, followed by odd fields for the second frame, and then alternating between even and odd.

The second approach generates a missing field feature map and fuses it with the feature map of the existing field [9]. In the method, the feature temporal interpolation module is used to generate a complete feature map. For the even field input, it first leverages F_{t-1}^{odd} and F_{t+1}^{odd} to produce F_t^{odd}, and then fuse F_t^{odd} with F_t^{even} to generate a complete feature map F_t^c.

The third method is known as vertical pixel shuffle. It divides a single field feature map in half on depth with the second half representing the missing field feature map [10], [8], which is illustrated in Fig. 3.10. In the figure, the feature map on the left represents the input field feature map and the feature map on the right represents the output. Vertical shuffle divides the input feature map into half on depth, the red one means the first half and the green one means the second. The two feature maps are combined together according to the figure to generate a complete frame feature map.

Chapter 4

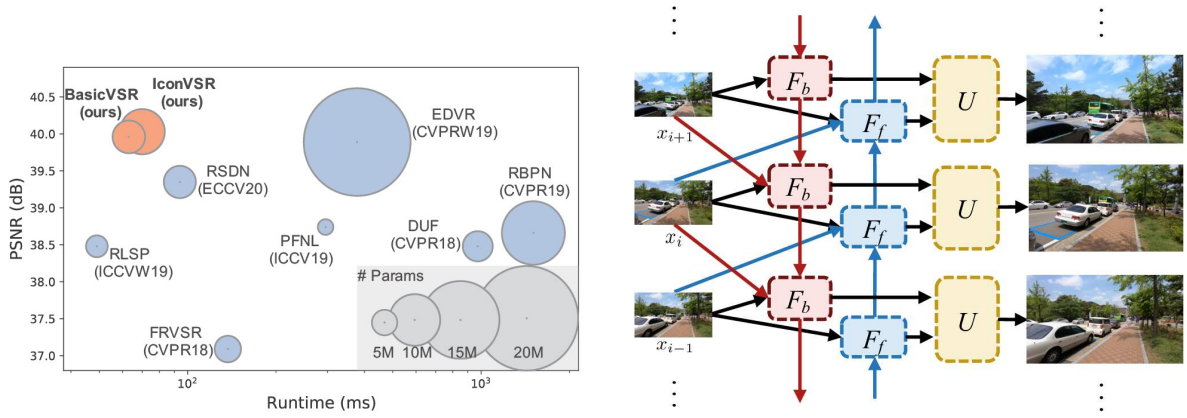
Methodology

Deinterlacing, similar to video super-resolution (VSR), leverages both spatial and temporal information to enhance video quality. However, because of more extensive research, there are more advanced and stable deep learning-based models for VSR in the literature. Among them, BasicVSR is a cutting-edge approach. It employs a bidirectional recurrent network that processes video frames in both forward and backward directions. This bidirectional framework captures long-range temporal data, utilizing information from both past and future frames for more accurate results. Therefore, we utilize BasicVSR as the backbone, introducing its effectiveness and simplicity to the deinterlacing task. Based on BasicVSR, we adjusted its structure for better adaptation to the deinterlacing task and proposed a novel reconstruction technique for deinterlacing.

4.1 BasicVSR

BasicVSR, introduced by Chan *et al.* in [16] is an advanced deep learning-based technique developed to address the challenges in video super-resolution (VSR). Unlike image super-resolution which focuses on a single image enhancement, VSR need to leverage the temporal continuity in video sequences to improve the resolution and quality of video frames. To address this challenge, BasicVSR combines the strengths of recurrent neural networks (RNNs) and convolutional neural networks (CNNs) to effectively capture spatial and temporal dependencies, thus producing high-quality video outputs.

Previous VSR methods often face challenges related to computational complexity and the need for sophisticated architectures to achieve high-quality results. In their paper,



(a) (b)

Figure 4.1: (a) Comparison of VSR methods. The size of the circles represents the number of parameters, the vertical axis represents the PSNR value, and the horizontal axis represents the model's running speed. (b) Structure of BasicVSR. The images on the left are the input, while the images on the right are the output. The red rectangles represent backward propagation, and the blue rectangles represent forward propagation. Both images are adapted from [16].

Chan *et al.* revisited previous VSR methods [17, 50] and decomposed them into four components, *Propagation*, *Alignment*, *Aggregation* and *Upsampling*. By analyzing and redesigning each previous component, they proposed BasicVSR, a simplified yet effective baseline model that balances performance with efficiency. The speed and performance comparison is illustrated in Fig. 4.1(a), which are tested on UDM10 dataset. The figure shows that BasicVSR achieves excellent results in a short amount of time and with fewer parameters, demonstrating its simplicity and efficiency.

At the core of BasicVSR is a bi-directional recurrent network, which consists of two main components: a forward and a backward propagation process. The network is illustrated in Fig. 4.1(b). The forward propagation processes frame sequentially from the beginning to the end of the video, while the backward propagation does the reverse. These two processes share the same network parameters, enabling the model to effectively learn from both directions. Additionally, the recurrent structure is designed to handle long-term dependencies, which are crucial for capturing motion information and fine details across frames. The network employs a combination of convolutional layers and recurrent units to achieve this, ensuring both spatial and temporal coherence in the output frames.

A notable strength of BasicVSR is its simplicity. By focusing on core principles and avoiding overly complex mechanisms, BasicVSR achieves impressive results in video super-resolution while maintaining a manageable level of computational overhead. This simplicity not only makes the model more accessible for practical applications but also sets a baseline for comparing more complex VSR models. The effectiveness of BasicVSR has been demonstrated through extensive experimentation on benchmark datasets, where it has shown competitive performance relative to more elaborate models.

In summary, BasicVSR represents a significant step forward in video super-resolution by offering a simple yet powerful baseline approach. Its design emphasizes efficiency and effectiveness, providing a solid foundation for future advancements in the field. The model's balance of performance and computational demands highlights its practical value, making it a noteworthy contribution to the ongoing development of video enhancement technologies.

4.2 Deinterlacing Network Structure

The proposed network, as depicted in Fig. 4.2, consists of three modules: *Spatial Extraction*, *Reconstruction*, and *Bidirectional Propagation*. Specifically, *Spatial Extraction* utilizes 5 residual blocks to extract spatial information from each individual field. *Reconstruction* restores the extracted feature maps to full frame size. Then, the reconstructed feature maps are split into two paths. One path, shown as dotted lines in Fig. 4.2, generates deinterlaced frames used to compute optical flow for the propagation module and add to the final output. The other path is input into *Bidirectional Propagation* to aggregate temporal information from highly related but misaligned frames. SPyNet is integrated within *Bidirectional Propagation* to estimate optical flow, which helps in aligning features across frames. After propagation, the feature maps from both directions are concatenated together to generate the output.

In most video super-resolution networks, the reconstruction module is at the end of the network. Nevertheless, in our model, it is placed between *Spatial Extraction* and *Bidirectional Propagation*. Because the edges in interlaced fields are not aligned and there is a disparity between odd and even fields, errors would occur when we utilize the temporal information without restoring them into full-frame size.

To evaluate the effectiveness of our proposed network structure, we compared it with BasicVSR-DI, where DI stands for deinterlacing. BasicVSR-DI has a BasicVSR structure with a deinterlacing reconstruction module at the end. It is important to note that the reconstruction module used in both models is identical, ensuring a fair comparison. The

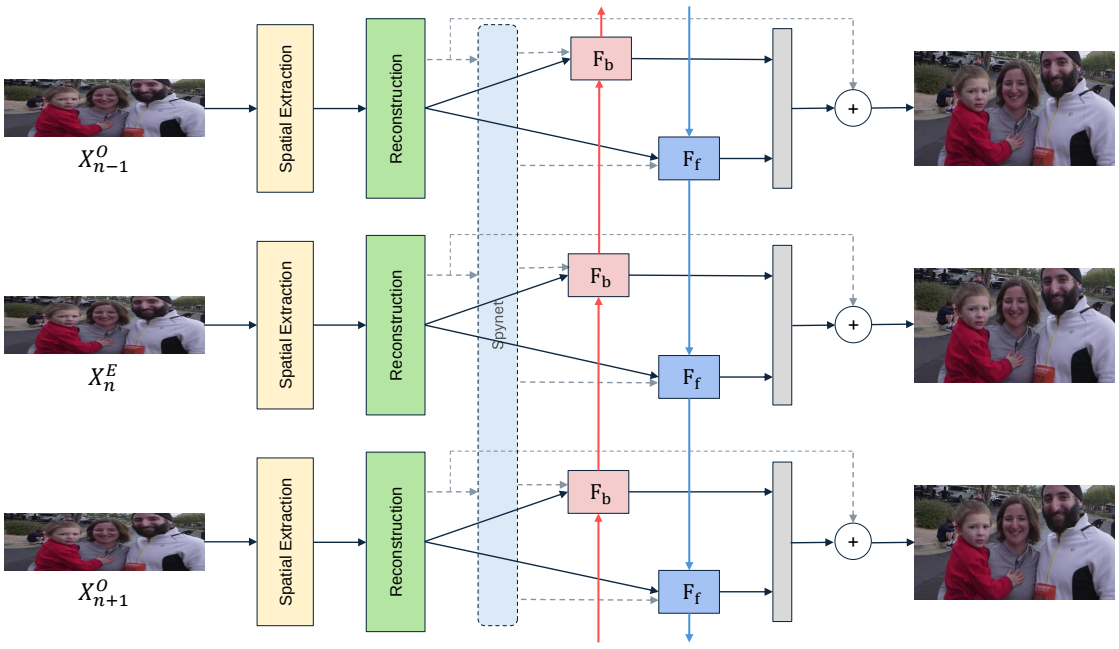


Figure 4.2: Network structure. The red colour represents backward propagation and the blue colour represents forward propagation. The gray colour represents concatenating feature maps from both directions together.

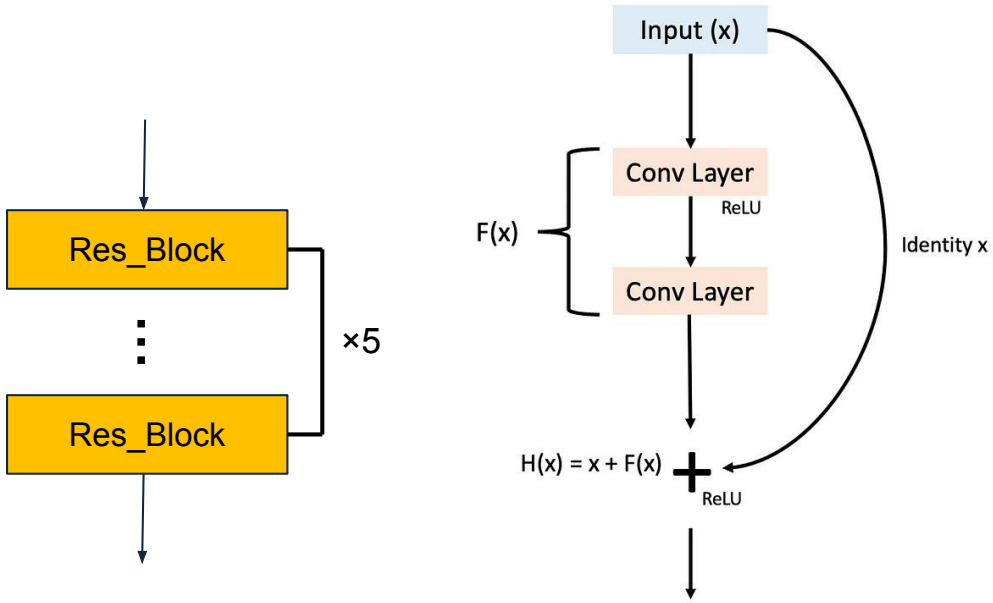
Table 4.1: The comparison between different *Reconstruction* positions (PSNR)

	Parameter(M)	**VimeoTest**	**Vid4**	**SPMC**	**UDM10**
BasicVSR-DI	6.37	46.27	34.06	47.91	46.20
Ours	6.55	47.68	35.80	49.11	46.80

performance comparison is illustrated in Table 4.1. The datasets in the table are illustrated in Section 5.3.1. Though both networks have a similar number of parameters, our proposed method performs considerably better on all testing datasets in terms of PSNR. A detailed explanation of PSNR is provided in Section 5.2.1.

4.2.1 Spatial Extraction

Spatial Extraction utilizes 5 residual blocks, which is a type of residual network structure. The specific structure is illustrated in Fig. 4.3(a). Residual network, or ResNet, was introduced by Kaiming He *et al.* [51]. Traditional deep networks often suffer from vanishing and exploding gradient problems, which hinder effective training as the depth of the network increases. ResNets tackle these issues through the use of residual connections or skip connections, which allow the network to learn residual mappings rather than the original, unreferenced functions. This is achieved by adding the input of each residual block to its output, thus creating a direct path for gradients to propagate and preserving the identity of the input. The specific structure is illustrated in Fig. 4.3(b).



(a) (b)

Figure 4.3: (a) Spatial Extraction module. (b) Residual block, from [12].

The architecture of ResNets consists of multiple residual blocks, each incorporating these skip connections, which facilitate the training of networks with hundreds or even thousands of layers. This design not only improves the training efficiency but also enhances generalization performance on various tasks, leading to state-of-the-art results in image recognition and beyond. Extensions of the original ResNet framework, such as ResNeXt

[52] and DenseNet [53], further build on these principles to optimize performance and computational efficiency. The innovative approach of ResNets has profoundly influenced subsequent developments in neural network design and continues to be a foundational element in modern deep learning methodologies.

4.2.2 SPyNet

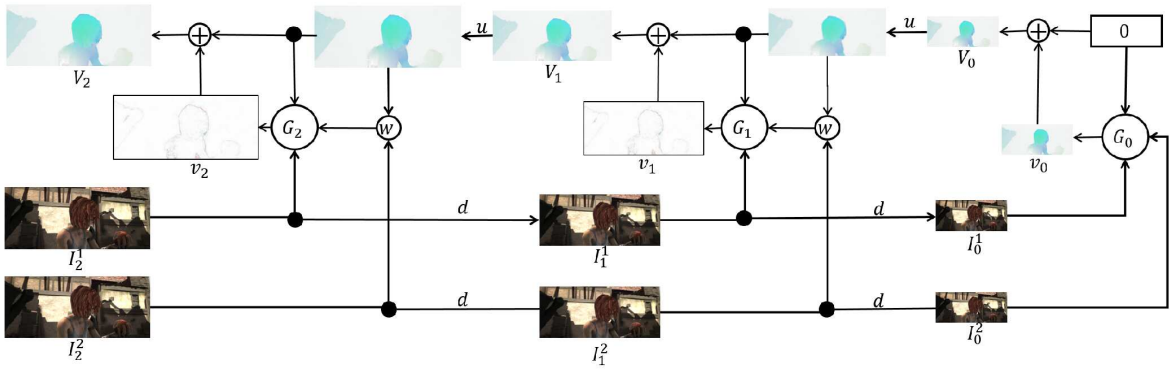


Figure 4.4: SPyNet, from [13].

In the model, SPyNet [13], also known as the Spatial Pyramid Network, generates optical flow for alignment by analyzing the motion of objects between consecutive frames. It estimates the displacement of pixels to correct for changes in viewpoint, enabling the alignment of images or video frames by compensating for shifts and rotations. This process helps ensure that images or video frames are registered accurately with each other.

One of the standout features of SPyNet is its use of pyramidal architecture. This design processes input images at multiple scales, enabling the system to handle varying levels of motion effectively and to gain a comprehensive understanding of the flow. This multi-scale approach contributes significantly to the algorithm's ability to manage different motion magnitudes and improves overall accuracy.

Despite its reliance on deep learning, SPyNet is engineered to be computationally efficient. This efficiency is crucial for applications that require real-time processing, such as video analysis or interactive systems. SPyNet has demonstrated strong performance in various benchmark datasets, showcasing its competitive edge in both accuracy and robustness compared to other state-of-the-art optical flow methods.

The applications of SPyNet are broad and impactful. In video analysis, it helps in tracking and understanding motion, which is valuable for fields like surveillance, autonomous driving, and sports analytics. In augmented reality (AR), accurate motion estimation is essential for creating seamless interactions between virtual and real elements. Additionally, in robotics, SPyNet's optical flow estimation assists in navigating and understanding the environment, which is vital for effective robot operation.

4.2.3 Bidirectional Propagation

Bidirectional propagation in neural networks is a concept that significantly enhances the ability of models to process and understand sequential data. It is a technique employed in various domains of machine learning, such as natural language processing (NLP), machine translation and financial forecasting, to enhance the flow and utilization of information across a network or model, addressing some of the limitations of traditional RNNs.

Traditional recurrent neural networks (RNNs) process sequences of data by maintaining a hidden state that updates as new data is received. While effective for many applications, RNNs can struggle with capturing long-range dependencies due to issues like vanishing or exploding gradients. Additionally, RNNs process data in a single direction, which limits their ability to capture contextual information from both the past and the future in a sequence.

Bidirectional RNNs were designed to overcome these limitations by processing sequences in both forward and backward directions. In a bidirectional RNN, one network processes the sequence from start to finish, while another network processes it from end to start. The outputs from these two RNNs are then combined, allowing the network to have access to information from both past and future contexts. This bidirectional approach enables the model to make more informed predictions or generate features that consider the entire sequence.

The bidirectional approach is particularly valuable in various applications. In natural language processing (NLP), for instance, understanding the context of words or phrases from both directions is crucial for tasks such as named entity recognition, machine translation, and sentiment analysis. Similarly, in speech recognition, considering the context of words in both directions improves accuracy. In time series analysis, bidirectional networks can leverage information from both past and future values to make better predictions.

In summary, bidirectional propagation enhances the capabilities of neural networks by allowing them to consider both past and future contexts in a sequence. This approach

leads to more robust and effective models for tasks involving sequential data, providing a richer understanding of the information contained in the sequence.

4.3 Reconstruction

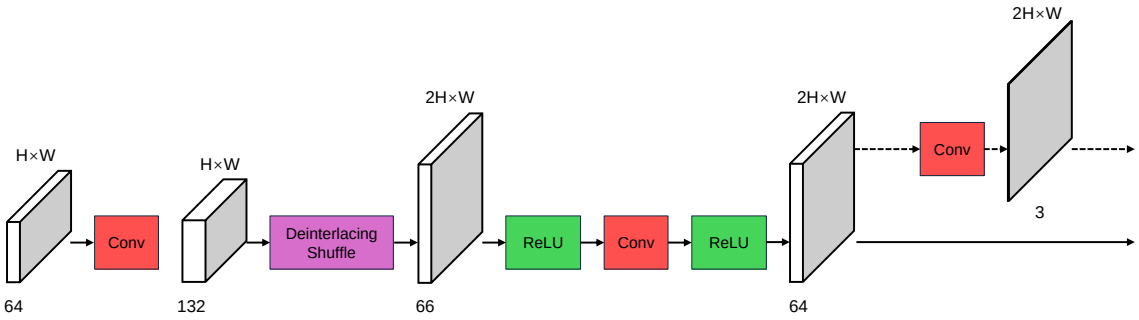


Figure 4.5: Reconstruction module.

Most existing deinterlacing models [6], [11], [7], [23] reconstruct the deinterlacing videos by predicting the missing fields and combining them with input fields. This type of method does not fully integrate input and output fields, and each time a fixed number of fields must be entered into the model. Another method [9] uses feature maps F_{t-1}^{odd} and F_{t+1}^{odd} to generate F_t^{odd}, then concatenate the upsampled F_t^{odd} with upsampled F_t^{even} to produce the complete feature map, which is sophisticated and time-consuming. Vertical pixel shuffle [10], [8] first splits a field feature map into two equal parts along the depth, then interleaves two feature maps together, where the second part represents the missing field feature map. In vertical pixel shuffle, features can only move in a vertical direction, which ignores the effect of other surrounding feature vectors. To present a simple and effective reconstruction method for video deinterlacing networks, we introduce a novel reconstruction method called deinterlacing shuffle. The technique inspired by pixel shuffle [24] is designed to restore each field feature map into a complete frame feature map by rearranging and stacking feature maps in a specific order.

Refer to Fig. 4.2, the *Spatial Extraction* module aggregates information from spatial dimensions, then transmits the features to the *Reconstruction* module, which recovers a complete frame feature map from the input by the deinterlacing shuffle. The comprehensive structure of *Reconstruction* is demonstrated in Fig. 4.5. First, the input feature map with

64 channels goes through a convolution layer to increase the channel number to 132, which is a multiple of 12. Then the deinterlacing shuffle restores the deinterlaced feature map from the interlaced input. After that, two rectified linear units (ReLU) and one convolution layer change the channel number of the deinterlaced feature map back to 64. The 64-channel feature map is then transmitted to the *Bidirectional Propagation*, as shown by a solid arrow in Fig. 4.5. In the meantime, as shown in dashed arrows, a deinterlaced frame is restored by a convolution layer from the 64-channel feature map and transmitted to SPyNet. The frame is used to compute optical flow in SPyNet and added to the final output.

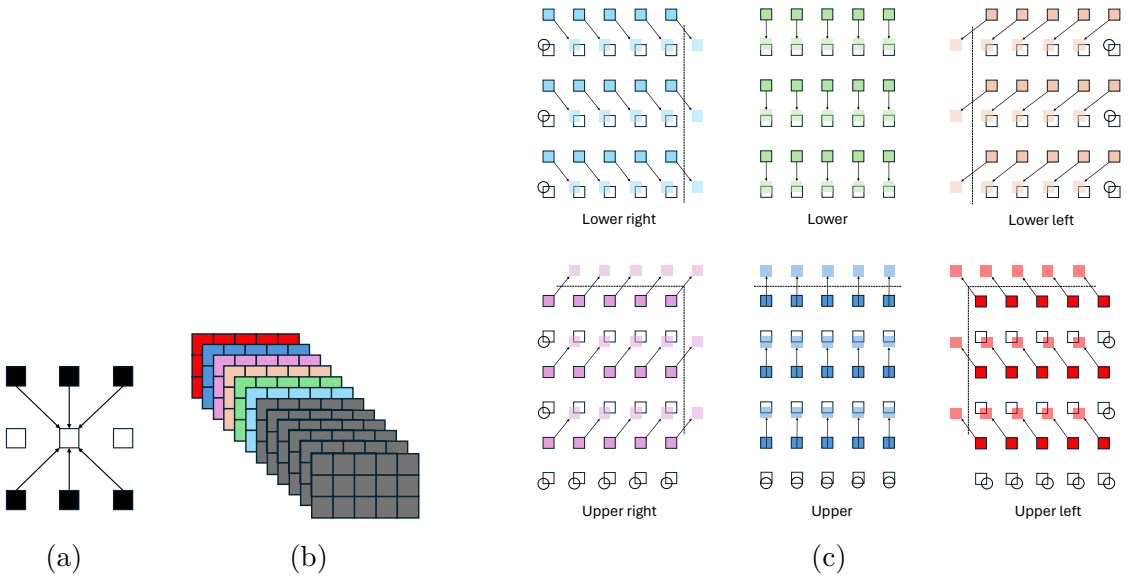


Figure 4.6: The operation of deinterlacing shuffle. Squares with different colours represent features with information. White squares represent features without information. White circles represent filled zeros. The dotted lines represent cut lines. (a) A feature vector to be restored receives features from the nearest six feature vectors. (b) An odd field feature map is divided into multiple groups. Each group contains 12 single-channel feature maps. The first 6 single-channel feature maps represent the input field feature map, and the last 6 maps shift to different directions to form the missing field feature map. (c) The shifted single-channel feature maps must be trimmed and filled. The last 6 single-channel feature maps from the odd field are shifted to compose the even field feature map.

The most important part in *Reconstruction* is the deinterlacing shuffle, which is a novel deinterlacing reconstruction technique introduced in this thesis. The deinterlacing shuffle restores every missing feature vector by distributing the nearest 6 feature vectors. All

surrounding feature vectors assign the same number of features to the missing feature vector. The operation is demonstrated in Fig. 4.6(a). In the figure, a feature vector with no information gets features from the nearest 6 feature vectors containing information. Suppose the size of each input feature map of the odd field to the deinterlacing shuffle is 3×5 (h×w), and the output of the deinterlacing shuffle is 6×5 (h×w). As shown in Fig. 4.6(b), an input feature map is divided into multiple groups, each consisting of 12 single-channel feature maps. Identical operations are applied to all groups. The first 6 single-channel feature maps, shown in gray colour, remain the same and compose the odd field feature map. The last 6 single-channel feature maps, represented by different colours, after some specific operations compose the even field feature map.

To obtain features from surrounding feature vectors, some shift operations must be applied to the last 6 single-channel feature maps. As shown in Fig. 4.6(c), different shift operations are employed to different single-channel feature maps represented by distinct colours. If we shift a single-channel feature map to the lower right direction, the features to be recovered get the features from the top left features. However, some features move outside the feature map, and some features that need to be restored do not receive features. To solve this problem, features moved outside are cut out and empty places are filled with zeros. Similar operations are applied to the single-channel feature maps shifting in other directions.

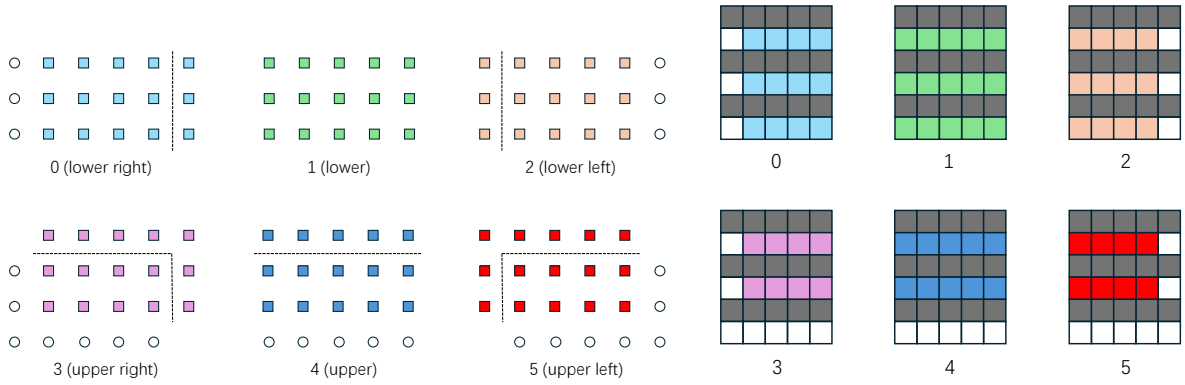


Figure 4.7: (a) The specific operation to the last 6 single-channel feature maps in each group. The 6 single-channel feature maps after operation can be concatenated together to compose the even field feature map. (b) The output of deinterlacing shuffle. The deinterlaced output is twice in height and a half in depth compared to the input.

Specific operations, illustrated in Fig. 4.7(a), are through shifting the last 6 single-channel feature maps in 6 directions to form the feature map of the missing field. The last 6 single-channel feature maps of 3×5 input are separated and numbered from 0 to 5. Different operations are taken based on the different directions of their movement. Based on their shifting directions, the features moved outside are cut off, and unfilled features are supplemented with zeros. As depicted in Fig. 4.7(b), after obtaining the feature map of the missing even field, the deinterlacing shuffle layer stitches the odd field feature map with the even field map together to obtain a full-frame feature map. If the input field is even, the first 6 single-channel feature maps represent the even field feature map, while the last 6 single-channel feature maps, after shifting, represent the odd field feature map.

Table 4.2: The comparison between different reconstruction methods (PSNR)

	VimeoTest	**Vid4**	**SPMC**	**UDM10**
Transposed	23.10	21.44	23.30	25.54
Vertical	28.78	26.53	27.87	29.28
Ours	29.93	27.25	28.88	30.53

After all the feature shifting operations above, a new group of single-channel feature maps with double size in height and a half size in depth compared to the input is created. We have compared our method with several other reconstruction methods. For instance, the "Transposed" method employs a transposed CNN to double the height of the input feature map; the "Vertical" method employs vertical pixel shuffle [10], [8] to divide a feature map in half on depth, with the second half representing the missing field feature map. Their results generated after *Reconstruction* are compared in Table 4.2. For the comparison, the reconstruction modules are located at the beginning of the model, and the PSNR values were generated right after the reconstruction modules. Our reconstruction method outperforms other approaches across all datasets.

4.4 Super-Resolution Version

Because of the versatility of the proposed model, we introduce an upsampling module to the network, thus implementing the super-resolution function, as shown in Fig. 4.8. The module is placed at the end of the network and employs the pixel shuffle technique. Pixel shuffle can enlarge the deinterlaced feature map to a higher spatial resolution. In addition, frames generated from reconstruction are upscaled using bilinear before adding

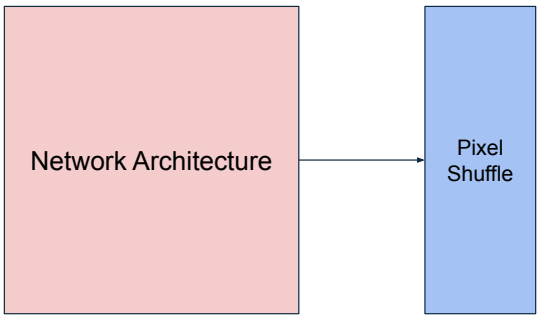


Figure 4.8: Super resolution version. The red box corresponds to the structure illustrated in Fig. 4.2.

to the final output. Hence, the model can fulfill both tasks. Deinterlacing converts the interlaced video to the progressive format, which helps eliminate the interlacing artifacts and produces smoother, more natural-looking video. Then, super-resolution enhances the resolution and quality of the deinterlaced video, making it visually sharper and more detailed. We train our models for upsampling factors $\times 2$ and $\times 4$. To prepare the training input, we first downsample the dataset using bicubic interpolation. Then, we apply the method described in Section 5.3.2 to generate the interlaced input. Their results are shown in Table 5.3. Completing both tasks through a single model allows for more comprehensive utilization of the input data, eliminating the need to convert interlaced video to progressive video before upscaling. Performing two separate processes would be more time-consuming and add complexity to operations.

4.4.1 Pixel Shuffle

The upsampling module in the super-resolution version contains pixel shuffle, which is a technique used in image processing and computer vision to enhance the spatial resolution of images. Originally introduced by Shi *et al.* [24], pixel shuffle is particularly effective in reconstructing high-resolution images from low-resolution inputs by rearranging pixels in a manner that increases spatial dimensions.

At its core, pixel shuffle operates by rearranging the channels of a feature map into spatial dimensions, as shown in Fig. 4.9. This technique involves a tensor reshaping operation where a high-dimensional feature map is split into smaller, spatially contiguous

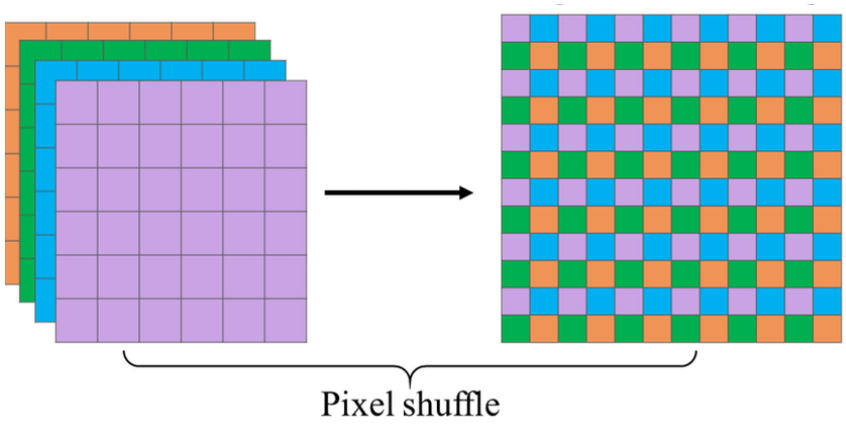


Figure 4.9: Pixel shuffle, from [14].

blocks. These blocks are then permuted to form a larger image. Specifically, the process takes a tensor of dimensions $(B, C \cdot r^2, H, W)$, where B is the batch size, C is the number of channels, r is the upscale factor, and (H, W) are the height and width of the feature map. The pixel shuffle operation redistributes these channels into a tensor of dimensions $B, C, H \cdot r, W \cdot r$, effectively increasing the image resolution by a factor of r.

The primary advantage of pixel shuffle lies in its ability to increase resolution with minimal computational overhead compared to traditional upsampling methods. Unlike interpolation-based methods that estimate missing pixel values, pixel shuffle rearranges existing data, which often results in sharper and more detailed images. This technique leverages the power of convolutional neural networks (CNNs) to first compress information into a lower-dimensional feature space and then use pixel shuffle to expand it back into a high-resolution image, thus maintaining more of the original detail and structure.

Chapter 5

Experiments

This chapter details the experimental setups, evaluation metrics, experiments and results, providing a clear view of the methodologies and findings essential to understanding the impact of our research on deep learning based deinterlacing.

5.1 Experimental Setups

5.1.1 Computational Resources and Libraries

We conducted experiments on Computer Canada's Narval cloud, which consists of 159 nodes, each equipped with 48 cores, 498 GB of memory, two AMD Milan 7413 CPUs, and four Nvidia A100 SXM4 GPUs. For each experiment, we allocated 6 CPU cores, 25 GB of memory, and one A100 GPU.

We employed the Python programming language and the PyTorch deep learning library, with the specific versions being torch=2.0.1 and torchvision=0.15.2. Additionally, we employed OpenCV for data preprocessing and for the post-processing of test results.

5.1.2 Training Settings

The model training configuration is as follows: GPU is assigned 6 workers and a batch size of 4. The dataset is enlarged by a ratio of 200. Training uses an exponential moving average decay of 0.999 and the Adam optimizer with a learning rate of 2e-4, weight decay

of 0, and betas of [0.9, 0.99]. A cosine annealing restart learning rate scheduler is employed, with periods of 300,000 iterations, restart weights of 1, and a minimum learning rate of 1e-7. The total number of iterations is 300,000, with no warm-up iterations. The flow is fixed at 5,000 iterations, and the learning rate multiplier for the flow is set to 0.125. For loss calculation, the charbonnier loss is used with a weight of 1.0 and mean reduction.

Next, we provide an explanation of several key aspects of the above configuration settings.

Exponential Moving Average (EMA)

EMA is a technique to smooth out data by giving more weight to recent observations while still considering older observations. In the context of deep learning, EMA is often used to keep a running average of model parameters (weights) during training. This helps in creating a more stable version of the model by reducing the volatility of parameter updates.

The EMA parameter controls how much weight is given to the previous average versus the current value. The formula for updating the EMA at step t is:

$$\text{EMA}_t = (1 - \text{ema\_decay}) \cdot x_t + \text{ema\_decay} \cdot \text{EMA}_{t-1}, \tag{5.1}$$

where,

- EMA_t is the updated exponential moving average at step t.
- x_t is the value at step t.
- ema_decay is a decay factor that determines how quickly the influence of past values decreases with its value ranging between 0 and 1.
- EMA_{t-1} is the previous exponential moving average.

A higher ema_decay value, close to 1, results in a smoother EMA that is more influenced by past values, providing stability but slower adjustment. Conversely, a lower ema_decay value, closer to 0, makes the EMA more responsive to recent parameter changes, resulting in a quicker adjustment to the current training state. This smoothing effect can improve the stability of training and often leads to better generalization of unseen data.

Adam Optimizer

The Adam optimizer, which stands for adaptive moment estimation, is a popular optimization algorithm used in training machine learning models, particularly deep neural networks. It combines ideas from two other optimization methods: AdaGrad and RMSProp.

Adam works by computing adaptive learning rates for each parameter. It maintains two separate moving averages for each parameter: the first moment (mean) and the second moment (uncentered variance). These moments are estimates of the gradient and the squared gradient, respectively. The algorithm updates these estimates at each iteration of training, which helps the optimizer adapt its learning rate to the parameters' gradients. This means that parameters with larger gradients get smaller updates and vice versa, leading to more efficient training.

In the Adam optimizer, the hyperparameters control how the optimization process adapts to the gradients during training. The learning rate (lr), set to 2×10^{-4}, determines the size of the steps taken in the direction of the gradient, with a smaller value ensuring more gradual updates to the parameters. The weight decay, which is set to 0 in this case, controls regularization by penalizing large weights to prevent overfitting, but no such penalty is applied here. The betas, with values $[0.9, 0.99]$, adjust the decay rates for the moving averages of the gradients and their squares: $\beta_1 = 0.9$ smooths the gradient estimates by considering recent gradients with some memory of past ones, while $\beta_2 = 0.99$ helps in adapting the learning rate based on the variance of the gradients by smoothing out fluctuations over time. These settings aim to balance between stable updates and efficient convergence during model training.

CosineAnnealingRestartLR

CosineAnnealingRestartLR is a sophisticated learning rate scheduling technique designed to enhance model training by dynamically adjusting the learning rate. This method initially starts with a high learning rate and decreases it following a cosine function, smoothing the reduction process. The learning rate decreases gradually to a minimum value, eta_min, which is set to a very small number like 1×10^{-7} in our configuration. This ensures that the learning rate never becomes too small, allowing for fine-tuning towards the end of each cycle.

One of the key features of CosineAnnealingRestartLR is its periodic nature, incorporating restarts into the scheduling process. In our setup, the learning rate follows a cosine annealing schedule over a specified period, which in this case is 300,000 steps. After this

period, the learning rate is reset to a higher value, and the cosine annealing process begins anew. The restart_weights parameter, set to 1 here, indicates that the learning rate is reset to the same initial value as before the restart. These periodic restarts enable the optimizer to escape local minima and explore the loss landscape more effectively.

By combining a smooth cosine decay with periodic restarts, CosineAnnealingRestartLR helps balance the exploration and exploitation of the optimization space, potentially leading to better convergence and improved model performance.

5.2 Evaluation Metrics

5.2.1 Peak Signal-to-Noise Ratio (PSNR)

Peak signal-to-noise ratio (PSNR) is a measure used to evaluate the quality of reconstructed signals, particularly in image and video processing. It quantifies the difference between the original video and the reconstructed version, providing a measure of the fidelity of the reconstructed video. PSNR is expressed in decibels (dB) and is derived from the mean squared error (MSE) between the original and the reconstructed video frames.

To calculate PSNR, we first compute the MSE, which measures the average squared differences between corresponding pixel values of the original and reconstructed frames. Let $I(x, y)$ and $K(x, y)$ represent the pixel values of the original and reconstructed images at coordinates (x, y), respectively. The MSE is given by:

$$\text{MSE} = \frac{1}{M \times N} \sum_{x=1}^{M} \sum_{y=1}^{N} [I(x, y) - K(x, y)]^2, \tag{5.2}$$

where M and N denote the height and width of the image.

The PSNR is then calculated using the formula:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}} \right). \tag{5.3}$$

Here, MAX_I represents the maximum possible pixel value, which is 255 for an 8-bit image.

Higher PSNR values indicate better quality and lower distortion in the reconstructed video. Typically, PSNR values above 30 dB are considered acceptable, and values over 40 dB indicate very high quality. However, while PSNR is useful, it doesn't always align perfectly with perceptual quality, as it only considers pixel-level differences and does not account for perceptual or structural factors.

5.2.2 Structural Similarity Index (SSIM)

Structural similarity index (SSIM) is a metric designed to assess the quality of video or image reconstruction by considering perceptual factors rather than just pixel-wise differences. Unlike traditional metrics such as peak signal-to-noise ratio (PSNR), SSIM focuses on how well the structural information in the images is preserved. This approach aligns more closely with human visual perception by evaluating luminance, contrast, and structural details.

To compute SSIM, the images are divided into overlapping windows or patches, and the index is calculated by comparing the local patterns of pixel intensities within these windows. The formula for SSIM between two images I (the original) and K (the reconstructed) is:

$$\text{SSIM}(I, K) = \frac{(2\mu_I\mu_K + C_1)(2\sigma_{IK} + C_2)}{(\mu_I^2 + \mu_K^2 + C_1)(\sigma_I^2 + \sigma_K^2 + C_2)}, \tag{5.4}$$

where,

- μ_I and μ_K are the average intensities of the windows from the original and reconstructed images, respectively.
- σ_I^2 and σ_K^2 are the variances of these windows.
- σ_{IK} is the covariance between the windows from the two images.
- C_1 and C_2 are constants used to stabilize the formula against weak denominators.

The SSIM index ranges from -1 to 1, where a value of 1 indicates that the two images are structurally identical, and values approaching -1 signify significant differences in structure. SSIM is generally preferred over metrics like MSE or PSNR because it more accurately reflects perceptual quality by taking into account the structural changes in luminance, contrast, and overall image structure.

In summary, SSIM offers a more perceptually relevant evaluation of video quality by assessing how well the structural details of the original image are preserved in the reconstructed version. It provides insights into the quality of the video by examining the preservation of luminance, contrast, and structural information, making it a valuable metric for quality assessment.

5.3 Experiments and Results

5.3.1 Datasets

We used Vimeo90k [54] as the training dataset. Vimeo90k is a substantial video dataset designed to advance research in video understanding. It comprises approximately 90,000 video clips, providing a rich and diverse collection of content for training and evaluating video analysis models. This dataset covers a wide range of categories and scenarios, making it valuable for developing models that can perform well across different types of video content. For testing datasets, we utilized VimeoTest, Vid4 [55], SPMC [56], and UDM10 [57].

In training, the training data is stored and accessed using the LMDB format. This format provides an efficient way to handle large-scale datasets by leveraging a key-value database structure, which facilitates fast read and write operations. LMDB, standing for lightning memory-mapped database, is particularly well-suited for handling large amounts of data, making it an effective choice for managing training data in machine learning workflows.

5.3.2 Interlacing Generation

Before training, we converted each video sequence in the training dataset into interlaced sequences as input data. Specifically, for each sequence, we retained only the odd rows for frames with odd indices (e.g., frames 1, 3, 5, 7) and only the even rows for frames with even indices (e.g., frames 2, 4, 6). The operation is shown in Fig. 5.1. The same operation is conducted on test datasets.

Figure 5.1: Generation of interlaced input. The numbers beside each frame are their indices, and odd or even lines are kept depending on the parity of their indices

5.3.3 Data Preprocessing

To prepare the data for model training, we employed several preprocessing techniques to enhance model robustness and generalization. Here are the details of the data preprocessing operations.

- **Random cropping**: We employed random cropping during data preprocessing to ensure that all images generated by the model are 128×128 pixels. Random cropping is a data augmentation technique used in deep learning to enhance model generalization. It involves selecting a random portion of an image and using that cropped segment to train the model. By choosing random coordinates and defining a fixed size for the crop, this technique ensures that different parts of the image are used in various training iterations. This approach exposes the model to diverse regions of the image, helping it learn more robust features and reducing the risk of overfitting.
- **Horizontal flipping**: We applied horizontal flipping as a data augmentation technique. Horizontal flipping refers to the process of flipping an image along its vertical axis, left to right. In other words, it creates a mirror image of the original image. Each image was randomly flipped along the vertical axis with a probability of 50%.

Horizontal flipping effectively doubles the size of your training dataset by creating new images from the existing ones. This augmentation strategy introduces additional variability and helps the model generalize better by exposing it to mirrored versions of the original images.

- **Sequence flipping**: For datasets consisting of image sequences, such as video frames, we also applied horizontal flipping to the entire sequence. Sequence flipping involves reversing the order of elements in a sequence. This can be thought of as creating a "mirror image" of the sequence along its temporal or sequential axis. It enables the model to learn features that are less dependent on sequence order, thus making it more adaptable and resilient.

- **Data standardization**: Data standardization is a preprocessing technique used to normalize the range of independent variables or features of data. It's especially important in deep learning to ensure that different features contribute equally to the model training and to improve the convergence speed and performance of the model. Data standardization transforms your data so that it has a mean of zero and a standard deviation of one. This is often done using the Z-score normalization formula:

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma}, \tag{5.5}$$

  where,

  - $x_{\text{standardized}}$ denotes the standardized value of x
  - x is the original data value.
  - μ represents the mean of the data.
  - σ is the standard deviation of the data.

  Standardized data can lead to more efficient training because it ensures that the gradients are on a similar scale. This can prevent issues where the optimizer might perform poorly due to different feature scales. Furthermore, features with larger ranges can dominate the learning process, making the model biased toward those features. Standardization helps in ensuring that the learning is fair and that all features are treated equally.

5.3.4 Comparisons with State-of-the-Art Methods

Due to the availability of data or source code, we can only compare our method with three other deep learning based deinterlacing methods, including DICNN [6], VDNet [9], and Gao [11]. Because deinterlacing is slightly similar to video spatio-temporal upscaling methods, we also compare our method with SOTA methods from this task, Zooming Slow-Mo [58] and TMNet [59] with simple modifications. The data of two video spatio-temporal upscaling methods was previously published in [9].

Table 5.1: Quantitative comparison (PSNR)

Method	**Parameters**	**VimeoTest** PSNR	**Vid4** PSNR	**SPMC** PSNR	**UDM10** PSNR
DICNN	0.06	40.46	31.32	37.94	41.11
Zooming Slow-Mo	11.77	45.61	34.59	47.10	44.67
TMNet	12.44	45.70	34.53	47.26	44.59
VDNet	9.23	46.45	34.83	47.84	45.52
Gao	8.88	46.50	35.46	48.19	46.20
Ours	6.55	47.68	35.80	49.11	46.80

Table 5.1 compares the PSNR, standing for peak signal-to-noise ratio, performance of four deinterlacing methods. Among these, our method achieves the highest PSNR scores across all tested datasets (VimeoTest, Vid4, SPMC, and UDM10), indicating superior performance in video quality. Specifically, our method surpasses Gao's method [11], the second best, by 1.18 dB on Vimeo90K and 0.92 dB on SPMC, demonstrating its effectiveness and consistency. In contrast, while VDNet and Gao also show strong performance, they do not reach the levels achieved by our method. And our network has fewer parameters than theirs, it has 2.33 million fewer parameters than Gao.

Table 5.2 compares the SSIM, standing for structural similarity index measure, performance of four deinterlacing methods. Our method demonstrates the highest SSIM scores in three of the four datasets, indicating superior structural similarity and quality. Gao follows closely with strong results. The only dataset where our method does not achieve the best result consists of videos with relatively static scenes, suggesting that our approach may not be optimal for recovering content in such scenarios.

Table 5.2: Quantitative comparison (SSIM)

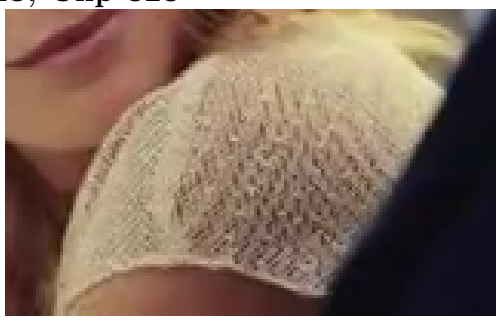
Method	**VimeoTest** SSIM	**Vid4** SSIM	**SPMC** SSIM	**UDM10** SSIM
DICNN	0.9791	0.9485	0.9795	0.9826
Zooming Slow-Mo	0.9909	0.9707	0.9959	0.9911
TMNet	0.9910	0.9698	0.9958	0.9912
VDNet	0.9922	0.9703	0.9965	0.9928
Gao	0.9935	0.9749	0.9972	0.9940
Ours	0.9939	0.9740	0.9973	0.9945

5.3.5 Visual Comparisons

In the following figures, Fig. 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, and 5.8, we present some visual comparisons of *Ours* with Yadif, DICNN [6] and BasicVSR-DI. Yadif standing for yet another deinterlacing filter, is a deinterlacing filter implemented in FFmpeg, BasicVSR-DI represents BasicVSR incorporating deinterlacing shuffle at the end, and GT means Ground Truth. We can only include methods with available implementations. As illustrated in the figures, while other methods struggle to generate images with fine details and delicate lines, *Ours* handles them well since our model uses reconstructed feature maps to leverage temporal information in the propagation module. In Fig. 5.6, Since the change between two consecutive frames within the boxed region is minimal, the difference between the input and the ground truth, which is formed by combining the odd and even rows of two consecutive frames, is also very small. Additionally, the PSNR of the cropped clips is shown near the images. Furthermore, we tested our model on some challenging test videos used in the video broadcasting industry and received excellent deinterlacing quality.

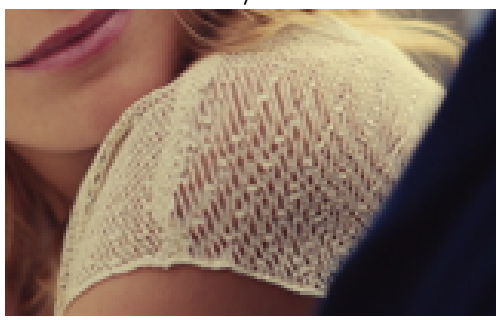
Sequence 0013, Clip 828

Input



Yadif/25.41

DICNN/26.19

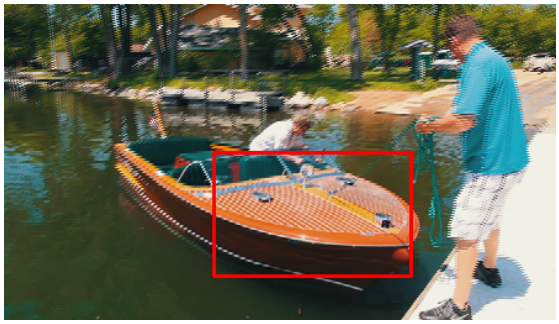


BasicVSR-DI/26.42

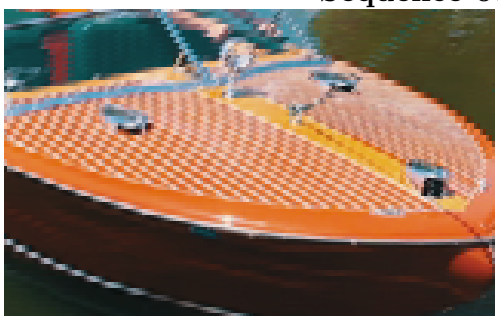
Ours/35.41

GT

Figure 5.2: Comparison of Sequence 0013, Clip 828 from VimeoTest



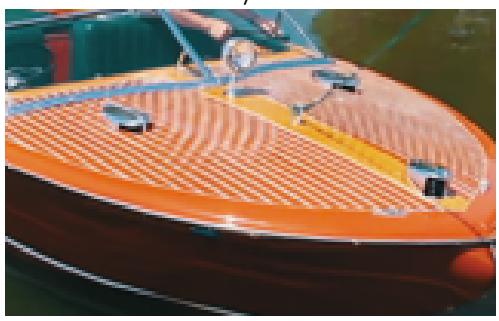
Sequence 0028, Clip 031



Input

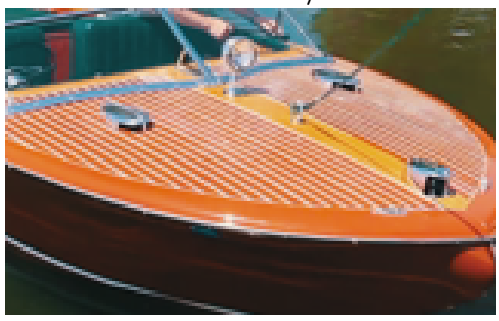
Yadif/22.30

DICNN/23.11



BasicVSR-DI/30.66

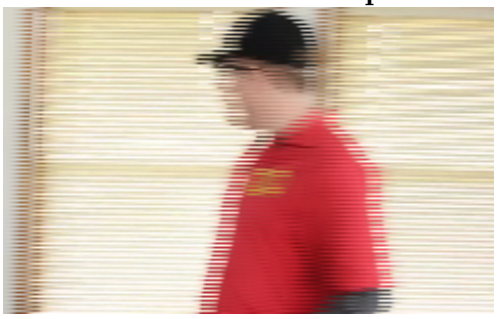
Ours/35.11



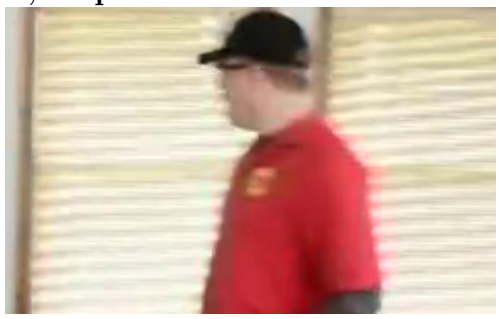
GT

Figure 5.3: Comparison of Sequence 0028, Clip 031 from VimeoTest

Sequence 0007, Clip 248



Input



Yadif/24.49

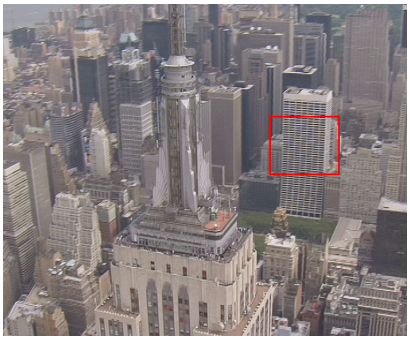
DICNN/23.75

BasicVSR-DI/29.15

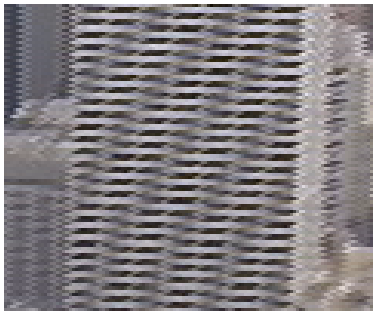
Ours/41.89

GT

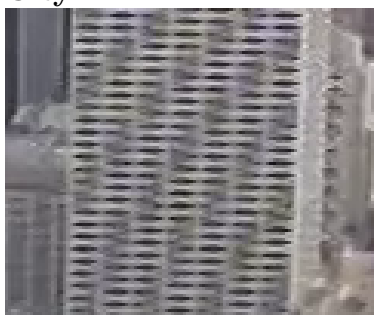
Figure 5.4: Comparison of Sequence 0007, Clip 248 from VimeoTest



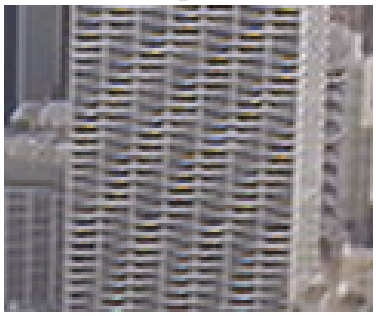
Vid4, City



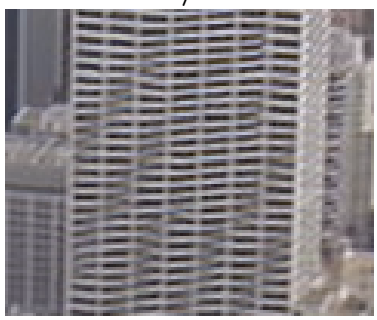
Input



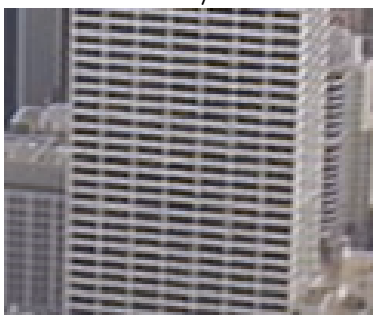
Yadif/19.50



DICNN/19.76



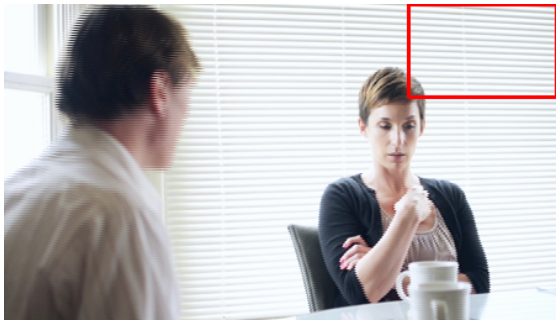
BasicVSR-DI/17.71



Ours/30.91

GT

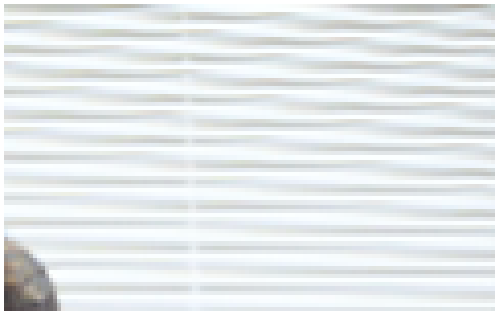
Figure 5.5: Comparison of City from Vid4



Sequence 0083, Clip 126

Input

Yadif/27.75



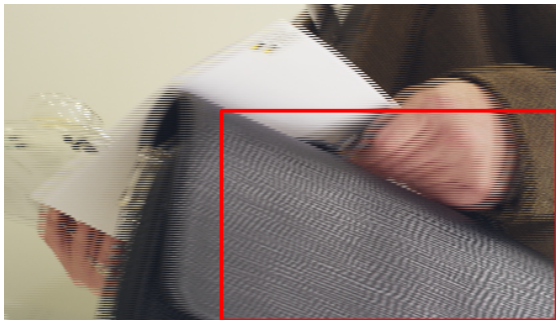
DICNN/27.70

BasicVSR-DI/26.68

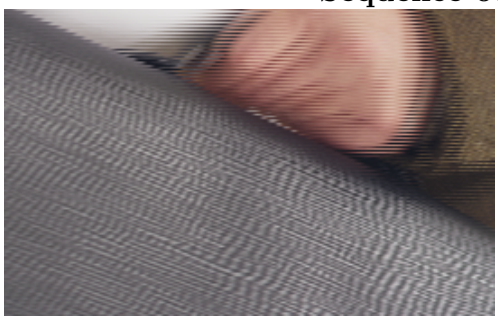
Ours/42.74

GT

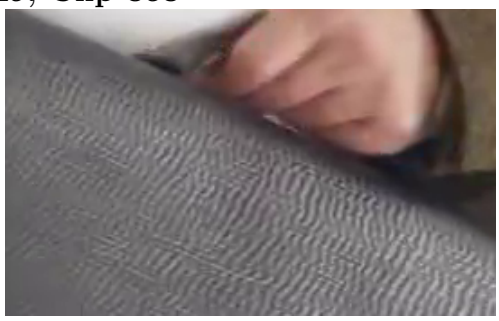
Figure 5.6: Comparison of Sequence 0083, Clip 126 from VimeoTest



Sequence 0049, Clip 858



Input



Yadif/24.53

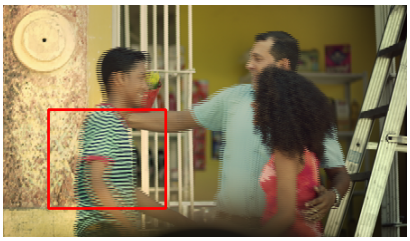
DICNN/24.39

BasicVSR-DI/26.74

Ours/37.40

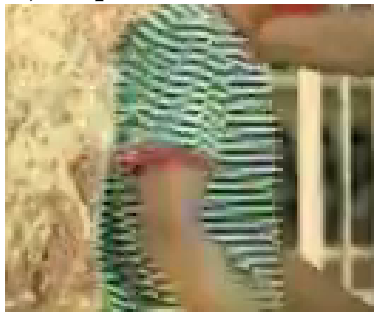
GT

Figure 5.7: Comparison of Sequence 0049, Clip 858 from VimeoTest



Sequence 0094, Clip 163

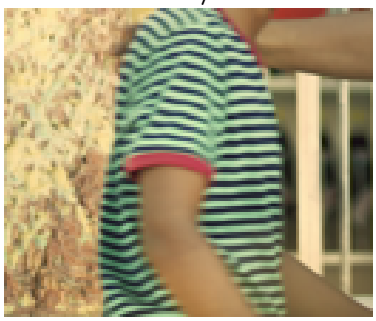
Input



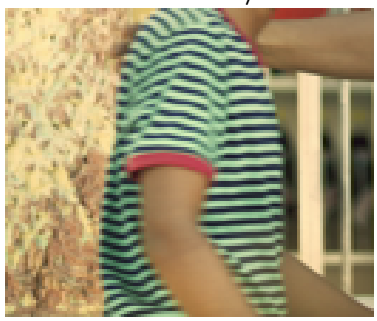
Yadif/21.52

DICNN/24.10

BasicVSR-DI/21.16



Ours/38.19



GT

Figure 5.8: Comparison of Sequence 0094, Clip 163 from VimeoTest

5.3.6 Experimental Results of Super-Resolution Version

Table 5.3: Quantitative comparison of super-resolution verison

Method		Upsampling ×2	Upsampling ×4
Parameters	Million	6.70	6.85
VimeoTest	PSRN	40.17	33.78
	SSIM	0.9753	0.9178
Vid4	PSRN	30.29	25.02
	SSIM	0.9386	0.7772
SPMC	PSRN	37.12	29.13
	SSIM	0.9715	0.8493
UDM10	PSRN	41.35	35.45
	SSIM	0.9809	0.9371

In this section, we present our evaluation data and visual results for the upsampling version of our model. In Table 5.3, *Upsampling ×2* and *Upsampling ×4* mean our method with upsampling scales of 2 and 4 respectively. Since our method is the first deinterlacing model with a super-resolution function, there is no comparison for them in Table 5.3.

In the following figures, Fig. 5.9, 5.10, and 5.11, we have shown some results of our upsampling models. In the figures, Interlaced_LR×2 means interlaced low-resolution images with 2× downscaling; Interlaced_LR×4 means interlaced low-resolution images with 4× downscaling; Upsample ×2 means output of *Upsampling ×2* model; Upsample ×4 means output of *Upsampling ×4* model.

GT

Interlaced_LR×2

Upsample×2

Interlaced_LR×4

Upsample×4

Figure 5.9: Results of Walk from Vid4

GT

Interlaced_LR×2

Upsample×2

Interlaced_LR×4

Upsample×4

Figure 5.10: Results of cact1_001 from SPMC

GT

Interlaced_LR×2

Upsample×2

Interlaced_LR×4

Upsample×4

Figure 5.11: Results of 004 from UDM10

Chapter 6

Conclusions and Future Work

We introduced a state-of-the-art deinterlacing model that enhances fine detail generation, overcoming the misalignment issues prevalent in previous methods. Our approach incorporates an innovative deinterlacing reconstruction technique and includes a super-resolution version of the model for better detail and clarity.

This work made the following contributions to the field:

- Based on the advanced video super-resolution model BasicVSR, we proposed a deep learning model for deinterlacing that bridges the gap between video super-resolution and deinterlacing. This approach leverages the efficiency and simplicity of video super-resolution models to enhance the deinterlacing process.
- We proposed a novel deinterlacing reconstruction method termed deinterlacing shuffle, which innovatively aggregates information from surrounding feature vectors to effectively reconstruct the missing parts of the image. This method enhances the deinterlacing process by leveraging surrounding pixel data to restore image quality with greater accuracy and detail.
- We position the reconstruction module before the propagation stage, allowing us to restore fields into frames before aggregating temporal information. This strategic placement helps to prevent misalignment between adjacent fields, ensuring that the temporal information is combined more accurately and consistently, ultimately improving the overall quality of the reconstructed frames.
- We proposed the first integrated model that simultaneously performs both deinterlacing and super-resolution tasks. This innovative approach not only streamlines the

processing pipeline, saving valuable time, but also maximizes the utilization of all available information in the input videos. By addressing both tasks within a single framework, our model enhances efficiency and effectiveness in video enhancement.

Our future work will focus on the real-time implementation of video deinterlacing by optimizing the use of available GPU resources. We aim to develop a solution that delivers high-quality deinterlacing performance while maintaining efficient processing speeds. This will involve refining our algorithms to balance computational demands with resource constraints, ultimately enabling practical and responsive applications in dynamic video environments.

References

[1] Intel Labs. Bringing parallelism to the web with river trail. https://intellabs.github.io/RiverTrail/tutorial/. Accessed: 2024-08-12.

[2] AILEPHANT. Recurrent neural network. https://ailephant.com/glossary/recurrent-neural-network/. Accessed: 2024-08-12.

[3] Gwanggil Jeon, Rafael Falcon, Luigi Gallo, Jechang Jeong, and Il Hong Suh. Single field deinterlacing scheme using edge-direction vectors in interlaced sequences. *Optical Engineering*, 48(6):067001–067001, 2009.

[4] Lejun Yu, Jintao Li, Yongdong Zhang, and Yanfei Shen. Motion adaptive deinterlacing with accurate motion detection and anti-aliasing interpolation filter. *IEEE Transactions on Consumer Electronics*, 52(2):712–717, 2006.

[5] Qian Huang, Debin Zhao, Siwei Ma, Wen Gao, and Huifang Sun. Deinterlacing using hierarchical motion analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 20(5):673–686, 2010.

[6] Haichao Zhu, Xueting Liu, Xiangyu Mao, and Tien-Tsin Wong. Real-time deep video deinterlacing. *arXiv preprint arXiv:1708.00187*, 2017.

[7] Yuqing Liu, Xinfeng Zhang, Shanshe Wang, Siwei Ma, and Wen Gao. Spatial-temporal correlation learning for real-time video deinterlacing. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021.

[8] Yang Zhao, Wei Jia, and Ronggang Wang. Rethinking deinterlacing for early interlaced videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(7):4872–4878, 2021.

[9] Yin-Chen Yeh, Jilyan Dy, Tai-Ming Huang, Yung-Yao Chen, and Kai-Lung Hua. VDNet: Video deinterlacing network based on coarse adaptive module and deformable recurrent residual network. *Neural Computing and Applications*, 34(15):12861–12874, 2022.

[10] Yang Zhao, Yanbo Ma, Yuan Chen, Wei Jia, Ronggang Wang, and Xiaoping Liu. Multiframe joint enhancement for early interlaced videos. *IEEE Transactions on Image Processing*, 31:6282–6294, 2022.

[11] Zhaowei Gao, Mingyang Song, Christopher Schroers, and Yang Zhang. Revitalizing legacy video content: Deinterlacing with bidirectional information propagation. *arXiv preprint arXiv:2310.19535*, 2023.

[12] Carla Martins. Understanding residual connections in neural networks. https://cdanielaam.medium.com/understanding-residual-connections-in-neural-networks-866b94f13a22. Accessed: 2024-08-12.

[13] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4161–4170, 2017.

[14] Mengjiao Qin, Sébastien Mavromatis, Linshu Hu, Feng Zhang, Renyi Liu, Jean Sequeira, and Zhenhong Du. Remote sensing single-image resolution improvement using a deep gradient-aware network with image-specific enhancement. *Remote Sensing*, 12(5):758, 2020.

[15] Marek Jancovic. Interlacing: The first video compression method. In *A Media Epigraphy of Video Compression: Reading Traces of Decay*, pages 77–119. Springer, 2023.

[16] Kelvin CK Chan, Xintao Wang, Ke Yu, Chao Dong, and Chen Change Loy. BasicVSR: The search for essential components in video super-resolution and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4947–4956, 2021.

[17] Xintao Wang, Kelvin CK Chan, Ke Yu, Chao Dong, and Chen Change Loy. EDVR: Video restoration with enhanced deformable convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[18] Wenbo Li, Xin Tao, Taian Guo, Lu Qi, Jiangbo Lu, and Jiaya Jia. MuCAN: Multi-correspondence aggregation network for video super-resolution. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*, pages 335–351. Springer, 2020.

[19] Yapeng Tian, Yulun Zhang, Yun Fu, and Chenliang Xu. TDAN: Temporally-deformable alignment network for video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3360–3369, 2020.

[20] Yan Huang, Wei Wang, and Liang Wang. Video super-resolution via bidirectional recurrent convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):1015–1028, 2017.

[21] Mehdi SM Sajjadi, Raviteja Vemulapalli, and Matthew Brown. Frame-recurrent video super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6626–6634, 2018.

[22] Takashi Isobe, Xu Jia, Shuhang Gu, Songjiang Li, Shengjin Wang, and Qi Tian. Video super-resolution with recurrent structure-detail network. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 645–660. Springer, 2020.

[23] Michael Bernasconi, Abdelaziz Djelouah, Sally Hattori, and Christopher Schroers. Deep deinterlacing. In *SMPTE Annual Technical Conf. Exhibition*, volume 3, 2020.

[24] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883, 2016.

[25] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[26] Ashish Vaswani. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[27] Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[31] F. Michaud, Chon Tam Le Dinh, and G. Lachiver. Fuzzy detection of edge-direction for video line doubling. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(3):539–542, 1997.

[32] Hoon Yoo and Jechang Jeong. Direction-oriented interpolation and its application to de-interlacing. *IEEE Transactions on Consumer Electronics*, 48(4):954–962, 2002.

[33] Min Kyu Park, Moon Gi Kang, Kichul Nam, and Sang Gun Oh. New edge dependent deinterlacing algorithm based on horizontal edge pattern. *IEEE Transactions on Consumer Electronics*, 49(4):1508–1512, 2003.

[34] Soonjong Jin, Wonki Kim, and Jechang Jeong. Fine directional de-interlacing algorithm using modified Sobel operation. *IEEE Transactions on Consumer Electronics*, 54(2):587–862, 2008.

[35] Sang-Jun Park, Gwanggil Jeon, and Jechang Jeong. Deinterlacing algorithm using edge direction from analysis of the DCT coefficient distribution. *IEEE Transactions on Consumer Electronics*, 55(3):1674–1681, 2009.

[36] Siyoung Yang, Donghyung Kim, and Jechang Jeong. Fine edge-preserving deinterlacing algorithm for progressive display. *IEEE Transactions on Consumer Electronics*, 55(3):1654–1662, 2009.

[37] Shyh-Feng Lin, Yu-Ling Chang, and Liang-Gee Chen. Motion adaptive interpolation with horizontal motion detection for deinterlacing. *IEEE Transactions on Consumer Electronics*, 49(4):1256–1265, 2003.

[38] Demin Wang, A. Vincent, and P. Blanchfield. Hybrid de-interlacing algorithm based on motion vector reliability. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(8):1019–1025, 2005.

[39] Yanfei Shen, Dongming Zhang, Yongdong Zhang, and Jintao Li. Motion adaptive deinterlacing of video data with texture detection. *IEEE Transactions on Consumer Electronics*, 52(4):1403–1408, 2006.

[40] Kwon Lee, Jonghwa Lee, and Chulhee Lee. Deinterlacing with motion adaptive vertical temporal filtering. *IEEE Transactions on Consumer Electronics*, 55(2):636–643, 2009.

[41] Piedad Brox, Iluminada Baturone, and Santiago Sanchez-Solano. Fuzzy motion-adaptive interpolation with picture repetition detection for deinterlacing. *IEEE Transactions on Instrumentation and Measurement*, 58(9):2952–2958, 2009.

[42] Hyunsoo Choi and Chulhee Lee. Motion adaptive deinterlacing with modular neural networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(6):844–849, 2011.

[43] Renxiang Li, Bing Zheng, and Ming L Liou. Reliable motion detection/compensation for interlaced sequences and its applications to deinterlacing. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(1):23–29, 2000.

[44] Yu-Lin Chang, Shyh-Feng Lin, Ching-Yeh Chen, and Liang-Gee Chen. Video deinterlacing by adaptive 4-field global/local motion compensated approach. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(12):1569–1582, 2005.

[45] Xinbo Gao, Juxia Gu, and Jie Li. De-interlacing algorithms based on motion compensation. *IEEE Transactions on Consumer Electronics*, 51(2):589–599, 2005.

[46] Hossein Mahvash Mohammadi, Pierre Langlois, and Yvon Savaria. A five-field motion compensated deinterlacing method based on vertical motion. *IEEE Transactions on Consumer Electronics*, 53(3):1117–1124, 2007.

[47] Min Li and Truong Nguyen. A de-interlacing algorithm using Markov random field model. *IEEE Transactions on Image Processing*, 16(11):2633–2648, 2007.

[48] Ying-Ru Chen and Shen-Chuan Tai. True motion-compensated de-interlacing algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(10):1489–1498, 2009.

[49] Yu-Cheng Fan and Chia-Hao Chung. De-interlacing algorithm using spatial-temporal correlation-assisted motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(7):932–944, 2009.

[50] Muhammad Haris, Gregory Shakhnarovich, and Norimichi Ukita. Recurrent back-projection network for video super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3897–3906, 2019.

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[52] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[53] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[54] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127:1106–1125, 2019.

[55] Ce Liu and Deqing Sun. On Bayesian adaptive video super resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(2):346–360, 2013.

[56] Xin Tao, Hongyun Gao, Renjie Liao, Jue Wang, and Jiaya Jia. Detail-revealing deep video super-resolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4472–4480, 2017.

[57] Peng Yi, Zhongyuan Wang, Kui Jiang, Junjun Jiang, and Jiayi Ma. Progressive fusion video super-resolution network via exploiting non-local spatio-temporal correlations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3106–3115, 2019.

[58] Xiaoyu Xiang, Yapeng Tian, Yulun Zhang, Yun Fu, Jan P Allebach, and Chenliang Xu. Zooming slow-mo: Fast and accurate one-stage space-time video super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3370–3379, 2020.

[59] Gang Xu, Jun Xu, Zhen Li, Liang Wang, Xing Sun, and Ming-Ming Cheng. Temporal modulation network for controllable space-time video super-resolution. In *Proceedings*

of the IEEE/CVF conference on computer vision and pattern recognition, pages 6388–6397, 2021.