|  |  |
| --- | --- |
|  | Moving Picture, Audio and Data Coding by Artificial Intelligencewww.mpai.community |

**Public document**

|  |  |
| --- | --- |
| **N1260** | 2023/06/14 |
| **Source** | Requirements (EEV) |
| **Title** | Trial specification of MPAI-EEV-0.4 WD0.1 |
| **Target** | MPAI-33 |

Table of Contents

[1.1 Scope 2](#_Toc137645088)

[1.2 Purpose 2](#_Toc137645089)

[1.3 Word usage 2](#_Toc137645090)

[2.Normative references 2](#_Toc137645091)

[3.Definitions, acronyms, and abbreviations 2](#_Toc137645092)

[3.1 Definitions 2](#_Toc137645093)

[3.2 Acronyms and abbreviations 2](#_Toc137645094)

[4.Conventions 3](#_Toc137645095)

[4.1 Overview 3](#_Toc137645096)

[4.2 Arithmetic Operators 3](#_Toc137645097)

[4.3 Relational Operators 3](#_Toc137645098)

[4.4 Assignment Operators 4](#_Toc137645099)

[4.5 Tensor Operators 4](#_Toc137645100)

[4.6 Description method of bitstream syntax 10](#_Toc137645101)

[5. Structure of Coded Bitstream 12](#_Toc137645102)

[6. Coding Mode Selection Syntax and Semantics 12](#_Toc137645103)

[7. Feature extraction 12](#_Toc137645104)

[7.1 Network Basics： 12](#_Toc137645105)

[Table x: Structure of Feature extraction network 13](#_Toc137645106)

[7.2 Function 13](#_Toc137645107)

[Table x: Function *forward* of the feature extraction network 13](#_Toc137645108)

[Table x: Usage of feature extraction module 13](#_Toc137645109)

[8. Intrinsic Motion 14](#_Toc137645110)

[8.1 Network Basics： 14](#_Toc137645111)

[Table x: structure of intrinsic motion module 14](#_Toc137645112)

[8.2 Function 15](#_Toc137645113)

[Table x: Usage of intrinsic motion module 15](#_Toc137645114)

[9. Compensatory Motion 16](#_Toc137645115)

[9.1 HiddenEncoder 16](#_Toc137645116)

[9.1.1 Network Basics： 16](#_Toc137645117)

[Table x: The structure of hidden encoder module 16](#_Toc137645118)

[9.2 quantize 17](#_Toc137645119)

[Table x: Definition of quantize function 17](#_Toc137645120)

[9.3 HiddenDecoder 18](#_Toc137645121)

[9.3.1 Network Basics： 18](#_Toc137645122)

[Table x: Structure of hidden decoder module 18](#_Toc137645123)

[9.4 Feature Aggregation 19](#_Toc137645124)

[9.4.1 Network Basics： 19](#_Toc137645125)

[Table x: Structure of feature aggregation module 19](#_Toc137645126)

[9.5 warp LSTM 20](#_Toc137645127)

[9.5.1 Network Basics： 20](#_Toc137645128)

[9.6 Function 20](#_Toc137645129)

[Table x: Usage of compensatory motion module 20](#_Toc137645130)

[10.2 Function 22](#_Toc137645131)

[Table x: Usage of feature restore module 22](#_Toc137645132)

[11. Inter Refine 22](#_Toc137645133)

[11.1 Network Basics： 22](#_Toc137645134)

[11.2 Function 23](#_Toc137645135)

[Table x: Usage of inter refine module 23](#_Toc137645136)

[12. Residue Coding 23](#_Toc137645137)

[12.1 Encoder 23](#_Toc137645138)

[12.1.1 Network Basics： 23](#_Toc137645139)

[Table x: Structure of encoder module. 24](#_Toc137645140)

[12.2 Quantize 24](#_Toc137645141)

[Table x: Structure of decoder module 25](#_Toc137645142)

[12.4.1 Network Basics： 26](#_Toc137645143)

[12.5 Function 26](#_Toc137645144)

[Table x: Usage of residue coding module 26](#_Toc137645145)

1.1 Scope

1.2 Purpose

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals is *required to*).

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals is *recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals is *permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals is **able to**).

The use of the word *must* is deprecated and cannot be used when stating mandatory requirements, *must* is used only to describe unavoidable situations.

The use of *will* is deprecated and cannot be used when stating mandatory requirements, *will* is only used in statements of fact.

2.Normative references

3.Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply.

Feature map: The output of a layer whose size is M×N×K. M and N are the spatial size. K is the number of channels.

EncResUnit:

nn.Sequential:

ConvLSTMCell:

b, c, h, w represent the batch size, the number of channels, height and width of the image

n\_c:

3.2 Acronyms and abbreviations

For the purpose of this document, the following abbreviations apply.

4.Conventions

4.1 Overview

The mathematical operators and priority levels used in this Standard are similar to those used in the C programming language. However, integer division and arithmetic shift operations are specifically defined. Unless otherwise stated, numbering and counting conventions generally begin from 0.

4.2 Arithmetic Operators

The following arithmetic operators are defined as Table 1.

Table 1 - Definitions of arithmetic operators

|  |  |
| --- | --- |
| Arithmetic operator | Definition |
| + | Additive operation |
| - | Subtraction (as a binary operator) or negation (as a unary prefix operator) |
| × | Multiplication |
| ab | Exponentiation. Specifies a to the power of b. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation. |
| / | Integer division with truncation of the result toward zero. For example, 7/4 and -7/-4 are truncated to 1, –7/4 and 7/-4 are truncated to -1. |
| ÷ | Used to denote division in mathematical equations where no truncation or rounding is intended. |

4.3 Relational Operators

The following relational operators are defined as Table 2.

Table 2 - Definitions of relational operators

|  |  |
| --- | --- |
| Relational operator | Definition |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| == | Equal to |
| != | Not equal to |

4.4 Assignment Operators

The following assignment operators are defined as Table 3.

Table 3 - Definitions of assignment operators

|  |  |
| --- | --- |
| Assignment operator | Definition |
| = | Assignment operator. |
| ++ | Increment, i.e., x++ is equivalent to x = x + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation. |
| -- | Decrement, i.e., x– – is equivalent to x = x – 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation. |
| += | Increment by amount specified, e.g., x + = 3 is equivalent to x = x + 3, and x + = (-3) is equivalent to x = x + (-3). |
| -= | Decrement by amount specified, e.g., x – = 3 is equivalent to x = x – 3, and x – = (-3) is equivalent to x = x – (-3). |

4.5 Tensor Operators

The following tensor operators are defined as Table 4.

Table 4 - Definitions of tensor operators

|  |  |
| --- | --- |
| Assignment operator | Definition |
| and | The definition of and shall follow Table 16 from the IEEE standard 2941. |
| argmax | The definition of argmax shall follow Table 17 from the IEEE standard 2941. |
| argmin | The definition of argmin shall follow Table 18 from the IEEE standard 2941. |
| argsort | The definition of argsort shall follow Table 19 from the IEEE standard 2941. |
| size | The definition of size shall follow Table 21 from the IEEE standard 2941. |
| average\_pool | The definition of average\_pool shall follow Table 24 from the IEEE standard 2941. |
| concat | The definition of concat shall follow Table 28 from the IEEE standard 2941. |
| constant | The definition of constant shall follow Table 29 from the IEEE standard 2941. |
| cond | The definition of cond shall follow Table 30 from the IEEE standard 2941. |
| conv2d | The definition of conv2d shall follow Table 31 from the IEEE standard 2941. |
| conv3d | The definition of conv3d shall follow Table 32 from the IEEE standard 2941. |
| conv2d\_transpose | The definition of conv2d\_transpose shall follow Table 33 from the IEEE standard 2941. |
| conv3d\_transpose | The definition of conv3d\_transpose shall follow Table 34 from the IEEE standard 2941. |
| crop | The definition of crop shall follow Table 38 from the IEEE standard 2941. |
| data | The definition of data shall follow Table 40 from the IEEE standard 2941. |
| embedding | The definition of embedding shall follow Table 46 from the IEEE standard 2941. |
| equal | The definition of equal shall follow Table 51 from the IEEE standard 2941. |
| expand | The definition of expand shall follow Table 53 from the IEEE standard 2941. |
| floor | The definition of floor shall follow Table 55 from the IEEE standard 2941. |
| less | The definition of less shall follow Table 61 from the IEEE standard 2941. |
| less\_equal | The definition of less\_equal shall follow Table 62 from the IEEE standard 2941. |
| is\_empty | The definition of is\_empty shall follow Table 63 from the IEEE standard 2941. |
| isfinite | The definition of isfinite shall follow Table 64 from the IEEE standard 2941. |
| log | The definition of log shall follow Table 68 from the IEEE standard 2941. |
| log\_loss | The definition of log\_loss shall follow Table 69 from the IEEE standard 2941. |
| lstm | The definition of lstm shall follow Table 71 from the IEEE standard 2941. |
| mat\_mul | The definition of mat\_mul shall follow Table 72 from the IEEE standard 2941. |
| max\_pool | The definition of max\_pool shall follow Table 73 from the IEEE standard 2941. |
| max\_roi\_pool | The definition of max\_roi\_pool shall follow Table 74 from the IEEE standard 2941. |
| mul | The definition of mul shall follow Table 75 from the IEEE standard 2941. |
| negative | The definition of negative shall follow Table 76 from the IEEE standard 2941. |
| not | The definition of not shall follow Table 77 from the IEEE standard 2941. |
| or | The definition of or shall follow Table 79 from the IEEE standard 2941. |
| pad | The definition of pad shall follow Table 80 from the IEEE standard 2941. |
| print | The definition of print shall follow Table 81 from the IEEE standard 2941. |
| pool2d | The definition of pool2d shall follow Table 83 from the IEEE standard 2941. |
| pool3d | The definition of pool3d shall follow Table 84 from the IEEE standard 2941. |
| pow | The definition of pow shall follow Table 85 from the IEEE standard 2941. |
| random\_normal | The definition of random\_normal shall follow Table 86 from the IEEE standard 2941. |
| random\_uniform | The definition of random\_uniform shall follow Table 87 from the IEEE standard 2941. |
| range | The definition of range shall follow Table 88 from the IEEE standard 2941. |
| reduce\_all | The definition of reduce\_all shall follow Table 89 from the IEEE standard 2941. |
| reduce\_any | The definition of reduce\_any shall follow Table 90 from the IEEE standard 2941. |
| reduce\_mean | The definition of reduce\_mean shall follow Table 91 from the IEEE standard 2941. |
| reduce\_max | The definition of reduce\_max shall follow Table 92 from the IEEE standard 2941. |
| reduce\_min | The definition of reduce\_min shall follow Table 93 from the IEEE standard 2941. |
| reduce\_prod | The definition of reduce\_prod shall follow Table 94 from the IEEE standard 2941. |
| reduce\_sum | The definition of reduce\_sum shall follow Table 95 from the IEEE standard 2941. |
| reduce\_varience | The definition of reduce\_varience shall follow Table 96 from the IEEE standard 2941. |
| relu | The definition of relu shall follow Table 97 from the IEEE standard 2941. |
| relu6 | The definition of relu6 shall follow Table 98 from the IEEE standard 2941. |
| reshape | The definition of reshape shall follow Table 99 from the IEEE standard 2941. |
| reverse | The definition of reverse shall follow Table 100 from the IEEE standard 2941. |
| rsqrt | The definition of rsqrt shall follow Table 101 from the IEEE standard 2941. |
| round | The definition of round shall follow Table 102 from the IEEE standard 2941. |
| shape | The definition of shape shall follow Table 103 from the IEEE standard 2941. |
| scatter | The definition of scatter shall follow Table 104 from the IEEE standard 2941. |
| sin | The definition of sin shall follow Table 105 from the IEEE standard 2941. |
| slice | The definition of slice shall follow Table 106 from the IEEE standard 2941. |
| softmax\_with\_cross\_entropy | The definition of softmax\_with\_cross\_entropy shall follow Table 107 from the IEEE standard 2941. |
| split | The definition of split shall follow Table 109 from the IEEE standard 2941. |
| space\_to\_depth | The definition of space\_to\_depth shall follow Table 110 from the IEEE standard 2941. |
| sqrt | The definition of sqrt shall follow Table 111 from the IEEE standard 2941. |
| square | The definition of square shall follow Table 112 from the IEEE standard 2941. |
| stack | The definition of stack shall follow Table 113 from the IEEE standard 2941. |
| sub | The definition of sub shall follow Table 114 from the IEEE standard 2941. |
| switch\_case | The definition of switch\_case shall follow Table 115 from the IEEE standard 2941. |
| topk | The definition of topk shall follow Table 116 from the IEEE standard 2941. |
| squeeze | The definition of squeeze shall follow Table 117 from the IEEE standard 2941. |
| transpose | The definition of transpose shall follow Table 118 from the IEEE standard 2941. |
| unsqueeze | The definition of unsqueeze shall follow Table 119 from the IEEE standard 2941. |
| unstack | The definition of unstack shall follow Table 120 from the IEEE standard 2941. |
| xor | The definition of xor shall follow Table 121 from the IEEE standard 2941. |
| scale | The definition of scale shall follow Table 122 from the IEEE standard 2941. |
| shift | The definition of shift shall follow Table 123 from the IEEE standard 2941. |
| while\_loop | The definition of while\_loop shall follow Table 124 from the IEEE standard 2941. |
| zeros | The definition of zeros shall follow Table 125 from the IEEE standard 2941. |
| full\_connected | The definition of full\_connected shall follow Table 126 from the IEEE standard 2941. |
| expand\_dims | The definition of expand\_dims shall follow Table 127 from the IEEE standard 2941. |
| batch\_normalization | The definition of batch\_normalization shall follow Table 128 from the IEEE standard 2941. |
| clip | The definition of clip shall follow Table 129 from the IEEE standard 2941. |
| elu | The definition of elu shall follow Table 130 from the IEEE standard 2941. |
| genm | The definition of genm shall follow Table 131 from the IEEE standard 2941. |
| hard\_sigmoid | The definition of hard\_sigmoid shall follow Table 132 from the IEEE standard 2941. |
| instance\_normalization | The definition of instance\_normalization shall follow Table 133 from the IEEE standard 2941. |
| log\_softmax | The definition of log\_softmax shall follow Table 134 from the IEEE standard 2941. |
| lrn | The definition of lrn shall follow Table 135 from the IEEE standard 2941. |
| leaky\_relu | The definition of leaky\_relu shall follow Table 136 from the IEEE standard 2941. |
| selu | The definition of selu shall follow Table 137 from the IEEE standard 2941. |
| prelu | The definition of prelu shall follow Table 138 from the IEEE standard 2941. |
| sigmoid | The definition of sigmoid shall follow Table 139 from the IEEE standard 2941. |
| sigmoid\_cross\_entropy\_with\_logits | The definition of sigmoid\_cross\_entropy\_with\_logits shall follow Table 140 from the IEEE standard 2941. |
| softmax | The definition of softmax shall follow Table 141 from the IEEE standard 2941. |
| softplus | The definition of softplus shall follow Table 142 from the IEEE standard 2941. |
| softsign | The definition of softsign shall follow Table 143 from the IEEE standard 2941. |
| tanh | The definition of tanh shall follow Table 144 from the IEEE standard 2941. |
| reduce\_log\_sum | The definition of reduce\_log\_sum shall follow Table 145 from the IEEE standard 2941. |
| reduce\_log\_sum\_exp | The definition of reduce\_log\_sum\_exp shall follow Table 147 from the IEEE standard 2941. |
| resize\_bilinear | The definition of resize\_bilinear shall follow Table 148 from the IEEE standard 2941. |
| resize\_nearest | The definition of resize\_nearest shall follow Table 149 from the IEEE standard 2941. |
| compute\_loss | The definition of compute\_loss shall follow Table 150 from the IEEE standard 2941. |
| apply\_optimize | The definition of apply\_optimize shall follow Table 151 from the IEEE standard 2941. |

4.6 Description method of bitstream syntax

The description method of bitstream syntax is similar to C language. Syntax elements in the bitstream are represented in bold type. Each syntax element is described by its name (all lower-case letters with underscored characters), syntax and semantics. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases, the syntax tables may use the values of other variables derived from syntax elements. Such variables appear in the syntax tables, or text, named by a mixture of lower-case letter and upper-case letter without underscoring. Variables starting with an upper-case letter are used for the decoding of the current and related syntax structures. Variables starting with an upper-case letter may also be used for the decoding of subsequent syntax structures. Variables starting with a lower-case letter are only used within the subclause from which they are derived.

The relationship between the mnemonic symbols of the syntax element value, the mnemonic symbols of the variable value and their values is described in the text. In some cases, the two are equivalent. A mnemonic symbol is represented by one or more groups of letters separated by underscores, each group starts with an upper-case letter and may contain more upper case letters.

When the length of the bit string is an integral multiple of 4, it can be represented by a hexadecimal notation. The prefix of hexadecimal notation is “0x”, for example, “0x1a” represents a bit string “0001 1010”.

In the conditional statement, 0 means FALSE, and non-zero means TRUE.

The syntax tables describe all the supersets of the bitstream syntax that conform to this Standard, and the additional syntax restrictions are described in the relevant clauses.

Table 5 shows an example of pseudo code describing syntax. It always means to read one data element from the bitstream whenever syntax elements are there.

Table 5 - example of pseudo code describing syntax

|  |
| --- |
| Pseudo code |
| /\* A statement is a descriptor of a syntax element, or indicates the existence, type, and value of a syntax element, two examples are given below. \*/ |
| syntax\_element |
| conditioning statement |
|  |
| /\* A group of statements enclosed in brackets is a compound statement and is treated functionally as a single statement.\*/ |
| { |
|  statement |
|  … |
| } |
|  |
| /\*A "while" structure tests whether the condition is TRUE. If TRUE, the statement is executed repeatedly until the condition is not TRUE. \*/ |
| while (condition) |
|  statement |
|  |
| /\*A “do … while” structure executes the statement once, and then tests the condition. It repeatedly executes the statement while the condition remains true. \*/ |
| do |
|  statement |
| while (condition) |
|  |
| /\*An “if … else” structure tests the condition first. If it is true, the primary statement is executed. Otherwise, the alternative statement is executed. If the alternative statement is unnecessary to be executed, the “else” part and corresponding alternative statement can be omitted. \*/ |
| if (condition) |
|  primary statement |
| else |
|  alternative statement |
|  |
| /\*A “for” structure executes the initial statement first, then tests the condition. If it is TRUE, the primary and subsequent statements are executed repeatedly until the condition is not TRUE.\*/ |
| for (initial statement; condition; subsequent statement) |
|  primary statement |
| /\*A “break” statement is used in do-while, while and for loop body, which can make the current loop body immediately terminate the loop. \*/ |
| break |

The parsing process and decoding process are described in text and pseudo code similar to C programming language.

5. Structure of Coded Bitstream

To be updated

6. Coding Mode Selection Syntax and Semantics

To be updated

7. Feature extraction

7.1 Network Basics：

* Parameters:
	+ n\_c: Number of layers of the network
* Input:
	+ One reference frame
		- A 4-D tensor of shape (b, c, h, w)
* Output:
	+ A list of multi-scale intermediate feature maps d1 and d2.
		- d1 – an array of shape (b, n\_c // 2, h // 2, w // 2)
		- d2 – an array of shape (b, n\_c // 2, h // 4, w // 4)

Table x shows the structure of feature extraction network.

Table x: Structure of Feature extraction network

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.conv1 = nn.Conv2d(3, self.n\_c, 4, 2, 1) |  |
| self.conv2 = nn.Sequential( EncResUnit(self.n\_c, self.n\_c, 1), EncResUnit(self.n\_c, self.n\_c, 1), nn.Conv2d(self.n\_c, self.n\_c, 4, 2, 1),) |  |

7.2 Function

The feature extraction module embeds each reference frame into the abstracted feature. Table x shows the forward propagation function of the feature extraction module.

Table x: Function *forward* of the feature extraction network

|  |  |
| --- | --- |
| **Definition of forward propagation function** | **Descriptor** |
| def forward(self, x): |  |
|  f1 = self.conv1(x) |  |
|  f2 = self.conv2(f1) |  |
|  return [f1, f2] |  |

After inputting a reference frame into the feature extraction module, the conv1 and conv2 modules will output multi-scale intermediate feature maps respectively.

Table x shows the usage of feature extraction module throughout the network.

Table x: Usage of feature extraction module

|  |  |
| --- | --- |
| **Usage** | **Descriptor** |
| seq\_len = ref\_list.size(1) | ref\_list: list of reference frames |
| for t in range(seq\_len): |  |
|  feature\_pool\_t = self.feature\_embedding(ref\_list[:, t]) |  |
|  h\_pool.append(feature\_pool\_t[-1]) |  |
|  feature\_pool.append(feature\_pool\_t) |  |
| h\_pool = torch.stack(h\_pool, dim = 1) | h\_pool: *f*ˆ*t*−1 |

feature\_pool\_t[0] and feature\_pool\_t[1] represents feature *d1* and *d2* respectively. The h\_pool stores *d2* as input for intrinsic motion modules. The feature\_pool stores *d1* and *d2* as input for the feature restoration and intrinsic motion modules.

8. Intrinsic Motion

8.1 Network Basics：

* Parameters:
	+ input\_dim: Number of channels in input
	+ hidden\_dim: Number of hidden channels
	+ kernel\_size: Size of kernel in convolutions
	+ num\_layers: Number of LSTM layers stacked on each other
	+ batch\_first: Whether or not dimension 0 is the batch or not
	+ bias: Bias or no bias in Convolution
	+ return\_all\_layers: Return the list of computations for all layers
* Input:
	+ Intermediate feature
		- A 5-D tensor either of shape (t, b, c // 2, h // 4, w // 4)
	+ Hidden state
		- Optional
* Output:
	+ A tuple of two lists of length num\_layers (or length 1 if return\_all\_layers is False).
		- 0 - layer\_output\_list is the list of lists of length t of each output
		- 1 - last\_state\_list is the list of last states
			* each element of the list is a tuple (h // 4, c // 2) for hidden state and memory
* Table x shows the structure of intrinsic motion module.

Table x: structure of intrinsic motion module

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| kernel\_size = self.\_extend\_for\_multilayer(kernel\_size, num\_layers) |  |
| hidden\_dim = self.\_extend\_for\_multilayer(hidden\_dim, num\_layers) |  |
| if not len(kernel\_size) == len(hidden\_dim) == num\_layers: |  |
| raise ValueError('Inconsistent list length.') |  |
| self.input\_dim = input\_dim |  |
| self.hidden\_dim = hidden\_dim |  |
| self.kernel\_size = kernel\_size |  |
| self.num\_layers = num\_layers |  |
| self.batch\_first = batch\_first |  |
| self.bias = bias |  |
| self.return\_all\_layers = return\_all\_layers |  |
| cell\_list = [] |  |
| for i in range(0, self.num\_layers): |  |
|  cur\_input\_dim = self.input\_dim if i == 0 else self.hidden\_dim[i - 1] |  |
| cell\_list.append(ConvLSTMCell(input\_dim=cur\_input\_dim, hidden\_dim=self.hidden\_dim[i], kernel\_size=self.kernel\_size[i], bias=self.bias, width = 16)) |  |
| self.cell\_list = nn.ModuleList(cell\_list) |  |

8.2 Function

The intrinsic motion module obtains the coarse temporal hypothesis *ft*1 in the feature space. Table x shows the usage of intrinsic motion module throughout the network.

Table x: Usage of intrinsic motion module

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| if pre\_state is not None: |  |
|  layer\_output, last\_state = self.conv\_lstm(h\_pool, pre\_state) | last\_state: *mi* |
| else: |  |
|  layer\_output, last\_state = self.conv\_lstm(h\_pool) |  |
| hidden\_feature = layer\_output[-1][:, t] | hidden\_feature: *ft*1 |

Due to the spatiotemporal correlation of the motion filed, the intrinsic motion can be implicitly inferred from the historical decoded frames by the ConvLSTM units. With the intrinsic motion *mit*−1, the temporal alignment is firstly implemented in the feature space. More specifically, the feature *f*ˆ*t*−1 extracted from the reference picture *x*ˆ*t*−1 is convolved with *mit*−1 by the ConvLSTM, producing the coarse inter prediction *ft*1 in the feature space. The temporal transition guided by the intrinsic motion can be written as

By the inference of ConvLSTM, we obtain the coarse temporal hypothesis *ft*1 in the feature space. Simultaneously, the intrinsic motion is updated along the temporal axis, which will be used at the next time step.

9. Compensatory Motion

9.1 HiddenEncoder

9.1.1 Network Basics：

* Parameters:
	+ in\_channels: Number of channels in input
	+ latent\_channels: Number of latent channels
	+ out\_channels: Number of channels in output
* Input:
	+ Reference frame and current frame
		- A 4-D tensor of shape (b, c \* 2, h, w) by *concat* the two frames
* Output:
	+ Latent feature
		- An array of shape (b, n\_c, h // 16, w // 16)
* Table x shows the structure of hidden encoder module.

Table x: The structure of hidden encoder module

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.\_model = nn.Sequential( nn.Conv2d(self.\_nic, self.\_nlc, 4, 2, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_nlc, 4, 2, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_nlc, 4, 2, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_noc, 4, 2, 1),) |  |

9.2 quantize

* Input:
	+ x: latent feature
	+ is\_training: true/false
	+ offset: 0
* Output:
	+ y: Quantified latent feature
* Table x shows the definition of quantize function.

Table x: Definition of quantize function

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| def quantize(x, is\_training, offset=0): |  |
|  if is\_training: |  |
|  y = QuantizeFunction.apply(x) |  |
|  else: |  |
|  y = torch.round(x - offset) + offset |  |
|  return y |  |

9.3 HiddenDecoder

9.3.1 Network Basics：

* Parameters:
	+ in\_channels: Number of channels in input
	+ latent\_channels: Number of latent channels
	+ out\_channels: Number of channels in output
* Input:
	+ Quantified latent feature
		- A 4-D tensor of shape (b, n\_c, h // 16, w // 16)
* Output:
	+ Decoded motion
		- An array of shape (b, n\_c, h // 4, w // 4)
* Table x shows the structure of hidden decoder module.

Table x: Structure of hidden decoder module

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.\_model = nn.Sequential( nn.ConvTranspose2d(self.\_nic, self.\_nlc, 4, 2, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), nn.ConvTranspose2d(self.\_nlc, self.\_nlc, 4, 2, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_nlc, 3, 1, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_noc, 3, 1, 1), ) |  |

9.4 Feature Aggregation

9.4.1 Network Basics：

* Parameters:
	+ in\_channels: Number of channels in input
	+ out\_channels: Number of channels in output
* Input:
	+ Coarse temporal hypothesis and decoded motion
		- A 4-D tensor of shape (b, n\_c \* 3 // 2, h // 4, w // 4) by concat above arrays
* Output:
	+ Compensatory motion
		- An array of shape (b, n\_c, h // 4, w // 4)
* Table x shows the structure of Feature Aggregation module.

Table x: Structure of feature aggregation module

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.combine = nn.Sequential( nn.Conv2d(self.\_ncs[0], self.\_ncs[1], 1), nn.ReLU(inplace=True), nn.Conv2d(self.\_ncs[1], self.\_ncs[2], 1), nn.ReLU(inplace=True), nn.Conv2d(self.\_ncs[2], self.\_ncs[3], 1)) |  |

9.5 warp LSTM

9.5.1 Network Basics：

* Input:
	+ Coarse temporal hypothesis and compensatory motion
* Output:
	+ Temporally aligned feature

See section 8.1 for more information on module structure, etc.

9.6 Function

Table x shows the usage of compensatory motion module throughout the network.

Table x: Usage of compensatory motion module

|  |  |
| --- | --- |
| **Usage** | **Descriptor** |
| mv\_feature = self.mv\_encoder(torch.cat([ref, x], 1)) |  |
| mv\_feature\_hat = quantize(mv\_feature, self.training) |  |
| mv\_hat = self.mv\_decoder(mv\_feature\_hat) |  |
| mv\_prior = self.feature\_agg(torch.cat([hidden\_feature, mv\_hat], 1)) | mv\_prior: *mct* |
| warp\_feature, \_ = self.warp\_lstm(hidden\_feature.unsqueeze(1),  [[mv\_prior[:, : self.n\_c // 2], mv\_prior[:, self.n\_c // 2 : ]]]) |  |

The intrinsic motion basically reflects the potential tendency of the moving objects. To further improve the performance of inter prediction, we provide the compensatory motion  to refine the coarse temporal hypothesis *ft*1 at the first step. As illustrated in Fig. 1, we combine the pair-wise reference picture *x*ˆ*t*−1 and current picture *xt* as input of the motion encoder side. Subsequently, the latent feature *yt* is produced through the non-linear transform layer, which is signaled as the side information. The decoded compensatory motion *mct* can be used to refine the temporal hypothesis *ft*1 with another ConvLSTM unit. As such, the motion compensation with *mct* can be written as

*ft*2*,* = *LSTM*(*ft*1*,mct*)

**10.** **Feature Restore**

**10.1 Network Basics：**

* Parameters:
	+ n\_c: Number of channels in input
* Input:
	+ Temporally aligned feature
		- A 4-D tensor of shape (b, n\_c // 2, h // 4, w // 4)
	+ d1
		- A 4-D tensor of shape (b, n\_c //2, h // 2, w // 2)
	+ d2
		- A 4-D tensor of shape (b, n\_c // 2, h // 4, w // 4)
* Output:
	+ Temporally aligned feature in the pixel space
		- An array of shape (b, c, h, w)
* Table x shows the structure of intrinsic motion module.

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| super().\_\_init\_\_() |  |
| self.n\_c = n\_c |  |
| self.deconv1 = nn.Sequential( nn.ConvTranspose2d(self.n\_c \* 2, self.n\_c, 4, 2, 1), DecResUnit(self.n\_c, self.n\_c, 1), DecResUnit(self.n\_c, self.n\_c, 1),) |  |
| self.deconv2 = nn.ConvTranspose2d(self.n\_c \* 2, 3, 4, 2, 1) |  |

10.2 Function

Table x shows the usage of feature restore module throughout the network.

Table x: Usage of feature restore module

|  |  |
| --- | --- |
| **Usage** | **Descriptor** |
| warp = self.feature\_restore(warp\_feature[-1][:, 0], feature\_pool[t][0], feature\_pool[t][1]) |  |

After the progressive temporal transition, we restore the temporally aligned feature *ft*2 into the pixel space as *x*˙*t*.

*x*˙*t* = *FR*(*ft*2*,d*2*,d*1).

11. Inter Refine

11.1 Network Basics：

* Parameters:
	+ in\_channels: Number of channels in input
	+ latent\_channels: Number of latent channels
	+ out\_channels: Number of channels in output
* Input:
	+ Temporally aligned feature in the pixel space and the reference frame
		- A 4-D tensor of shape (b, c \* 2, h, w) by concat the 2 arrays
* Output:
	+ Inter prediction
		- An array of shape (b, c, h, w)
* Table x shows the structure of intrinsic motion module.

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.\_model = nn.Sequential( nn.Conv2d(self.\_nic, self.\_nlc, 3, 1, 1), nn.PReLU(), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_nlc, 3, 1, 1), nn.PReLU(), nn.Conv2d(self.\_nlc, self.\_noc, 1),) |  |

11.2 Function

Table x shows the usage of inter refine module throughout the network.

Table x: Usage of inter refine module

|  |  |
| --- | --- |
| **Usage** | **Descriptor** |
| pred = self.inter\_refine(torch.cat([ref, warp], 1)) + warp |  |

We propose a spatiotemporal enhancement network to obtain the inter prediction *x*¯*t*. The reference picture *x*ˆ*t*−1 is used as guidance to refine the inter prediction.

*x*¯*t* = *IR*(*x*˙*t, x*ˆ*t*−1)

12. Residue Coding

12.1 Encoder

12.1.1 Network Basics：

* Parameters:
	+ in\_channels: Number of channels in input
	+ latent\_channels: Number of latent channels
	+ out\_channels: Number of channels in output
* Input:
	+ Residue
		- A 4-D tensor of shape (b, c, h, w)
		- The difference between the original *xt* and prediction *x*¯*t*
* Output:
	+ Latent residue
		- An array of shape (b, n\_c, h //16, w // 16)
* Table x shows the structure of encoder module.

Table x: Structure of encoder module.

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.\_model = nn.Sequential( nn.Conv2d(self.\_nic, self.\_nlc, 4, 2, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_nlc, 4, 2, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_nlc, 4, 2, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), EncResUnit(self.\_nlc, self.\_nlc, 1), nn.Conv2d(self.\_nlc, self.\_noc, 4, 2, 1),) |  |

12.2 Quantize

* Input:
	+ y: Latent residue
	+ is\_training: true/false
	+ offset: 0
* Output:
	+ y\_hat: Quantified latent residue

See section 9.2 for details

**12.3 Decoder**

**12.3.1 Network Basics：**

Parameters:

* + in\_channels: Number of channels in input
	+ latent\_channels: Number of latent channels
	+ out\_channels: Number of channels in output
* Input:
	+ Quantified latent residue
		- A 4-D tensor of shape (b, n\_c, h //16, w // 16)
* Output:
	+ Reconstruction frame
		- An array of shape (b, c, h, w)
* Table x shows the structure of decoder module.

Table x: Structure of decoder module

|  |  |
| --- | --- |
| **Network Structure** | **Descriptor** |
| self.\_model = nn.Sequential( nn.ConvTranspose2d(self.\_nic, self.\_nlc, 4, 2, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), nn.ConvTranspose2d(self.\_nlc, self.\_nlc, 4, 2, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), nn.ConvTranspose2d(self.\_nlc, self.\_nlc, 4, 2, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), DecResUnit(self.\_nlc, self.\_nlc, 1), nn.ConvTranspose2d(self.\_nlc, self.\_noc, 4, 2, 1),) |  |

**12.4 loop\_filter**

12.4.1 Network Basics：

* Input:
	+ Coarse reconstructed frame and reference frame
		- A 4-D tensor of shape (b, c \* 2, h, w) by concat the 2 arrays
* Output:
	+ Final reconstruction frame
		- An array of shape (b, c, h, w)

See section 11.1 for more details

12.5 Function

Table x shows the usage of residue coding module throughout the network.

Table x: Usage of residue coding module

|  |  |
| --- | --- |
| **Usage** | **Descriptor** |
| resi = x - pred |  |
| y = self.encoder(resi) |  |
| y\_hat = quantize(y, self.training) |  |
| resi\_hat = self.decoder(y\_hat) |  |
| x\_hat = resi\_hat + pred |  |
| x\_refine = self.loop\_filter(torch.cat([ref, x\_hat], 1)) + x\_hat |  |

Subsequently, we introduce the variational auto-encoder based image compression with hyperprior entropy module to compress the residue. With the reconstructed residue *r*ˆ*t* and prediction *x*¯*t*, the reconstruction *x*¨*t* is derived.

*x*¨*t* = ¯*xt* + *r*ˆ*t*.

In order to remove the compression artifacts, we place the inloop filter network at the end of whole framework. Through the in-loop filtering network, we obtain the reconstruction *x*ˆ*t*. *x*ˆ*t* is appended into the decoded picture buffer (DPB) for the subsequent inter coding.

*x*ˆ*t* = *LF*(*x*¨*t, x*ˆ*t*−1).