Summary for Jingtun ZHANG's 2019 Summer Intern @University of California Santa Barbara

Jingtun ZHANG

16th September, 2019

Contents

1 Introduction

This is a summary of Jingtun ZHANG's 2019 summer intern @University of California Santa Barbara

Paper reading note can be found at Github link

Studying note can be found at Github link

Weekly report can be found at Github link

2 Graph Neural Network Survey

2.1 Background

2.1.1 Origin GNN

Target problem: learn a state embedding $\mathbf{h}_v \in \mathbb{R}^s$ for each node

Traditional procedure:

$\mathbf{h}_v = f(\mathbf{x}_v, edge\_attr_v, \mathbf{h}_u, \mathbf{x}_u)$
u means neighbors of v, f is local parameterized transition function
$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v)$
g is local output function

Typical loss:

$loss = \sum_{i=1}^{p}(target_i - output_i)$

Details can be found at Github link

2.1.2 From spectral to propagation: Graph Convolutional Network (GCN)

$g_\theta(\Lambda)$ can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order:

$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\widetilde{\Lambda})$

1. $\widetilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N$, λ_{max} denotes the largest eigenvalue of L
2. $\theta' \in \mathbb{R}^K$: vector of Chebyshev coefficients
3. $Chebyshev \quad polynomial : T_0(x) = 1, \quad T_1(x) = x, \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$
4. SO: $g_{\theta'} \star x \approx U \sum_{k=0}^{K} \theta'_k T_k(\widetilde{\Lambda}) U^\top x = \sum_{k=0}^{K} \theta'_k T_k(\widetilde{L}) x$
5. $\widetilde{L} = \frac{2}{\lambda_{max}} L - I_N$
6. $K^{th} - order$ polynomial in the Laplacian: it depends only on nodes that are at maximum K steps away from the cantral node
7. with $K = 1$ and $\lambda_{max} \approx 2$:
   $g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$
   with $\theta = \theta'_0 = -\theta'_1$ and renormalization trick $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}$
   from C input channelsand F filters:
   $Z = \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} X \Theta$: complexity:$\mathcal{O}(|\varepsilon|FC)$
8. $\Theta \in \mathbb{R}^{C \times F}$: matrix of filter parameters
9. $Z \in \mathbb{R}^{N \times F}$: convolved signal matrix

Layer-wise propagation rule:

$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})$

$\widetilde{A} = A + I_N$: adjacency matrix with added self connection

$\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$:

$W^{(l)}$: layer-specific trainable weight matrix

$\sigma(\cdot)$: activation function

$H^{(l)} \in \mathbb{R}^{N \times D}$: the matirx of activations in the l^{th} layer, $H^{(0)} = X$

Details can be found at Github link

2.1.3 Why GNN from CNN

1. Graphs are the most typical locally connected structure
2. Share weights reduce the computational cost compared with traditional spectral graph theory
3. Multilayer structure is the key to deal with hierarchical patterns, which captures the featuofres of various sizes
4. CNNs or RNNs need a specific order, but there is no natural order of nodes in graph, GNNs output is input order invarient
5. Human intelligencce is most based on the graph, GNNs can do information propagation guided by the graph structure
6. GNNs explores to generate the graph from non-structural data

2.2 Models

- GDyNet and CGCNN model: Application of GNN in materials. Details can be found at Github link

2.3 System View Optimization

- Tigr

  Transform **i**rregular **g**raphs into more **r**egular ones such that the graphs can be processed more efficiently on GPU-like architectures while guaranteeing correctness.

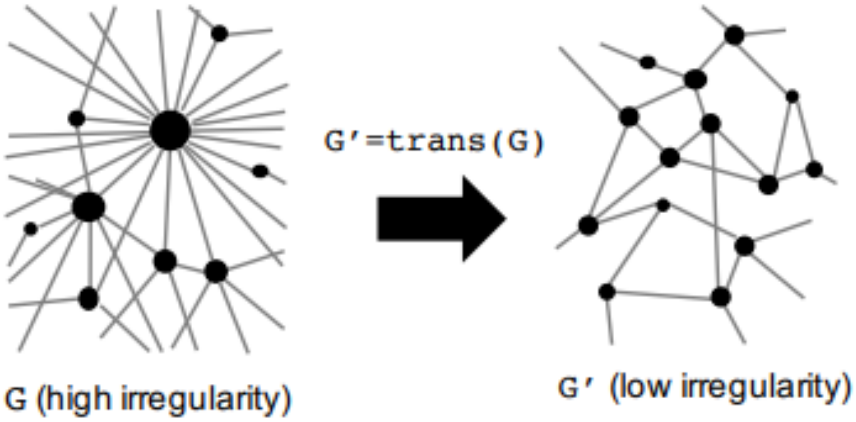


Figure 1. Illustration of Graph Irregularity Reduction.

  Details can be found at Github link

- Fast train GNN on dense hardware

  Permute nodes to expose low bandwidth structure and express GNN propagation in terms of application of dense matrix multiply primitive.

  Details can be found at Github link

2.4 General Model Project: PyTorch Geometric

Github Project rusty1s/pytorch_geometric implements many important GNN models with general GNN model Message Passing Neural Network, and builds an end-to-end graph data loading to testing model architecture. Detail studying note can be found at Github Link

I modified this project for the following research: Code can be found at Github Link

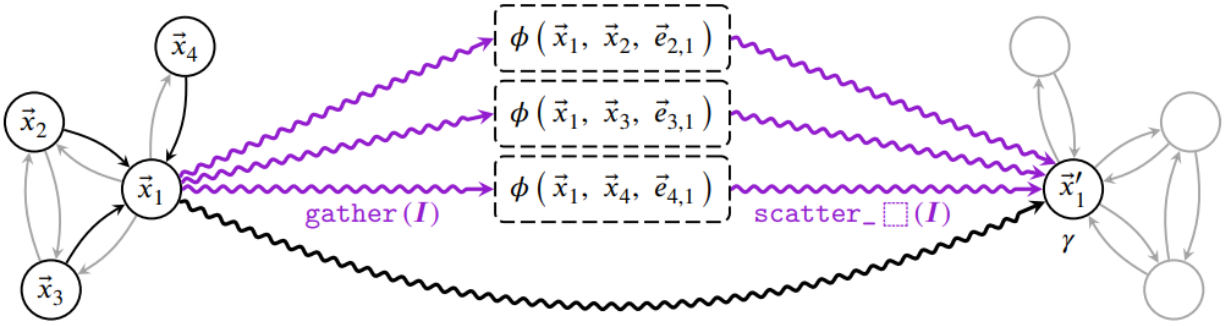


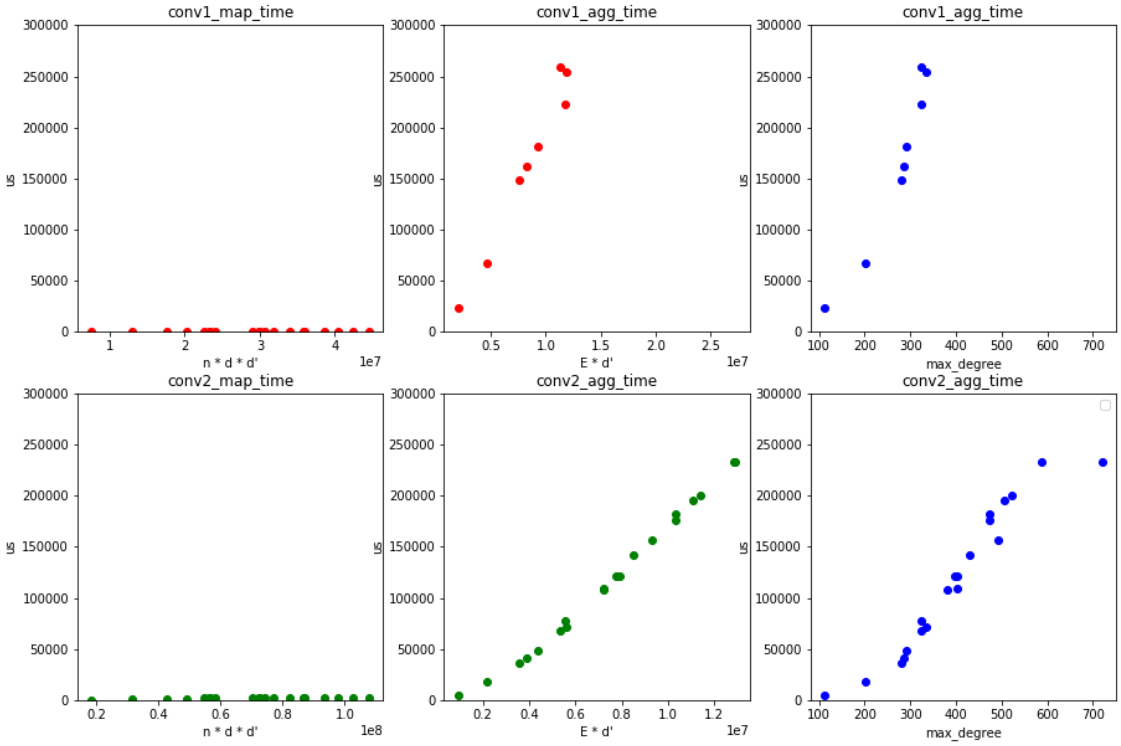
Figure 1: Computation scheme of a GNN layer by leveraging gather and scatter methods based on edge indices $\boldsymbol{I}$, hence alternating between node parallel space and edge parallel space.

Figure 1:

- Profiling of GNN models

  Complexity analysis of MPNN network

  Details can be found at Github link



- Visualization of Pooling effectiveness

  In topology domain: **Less nodes**

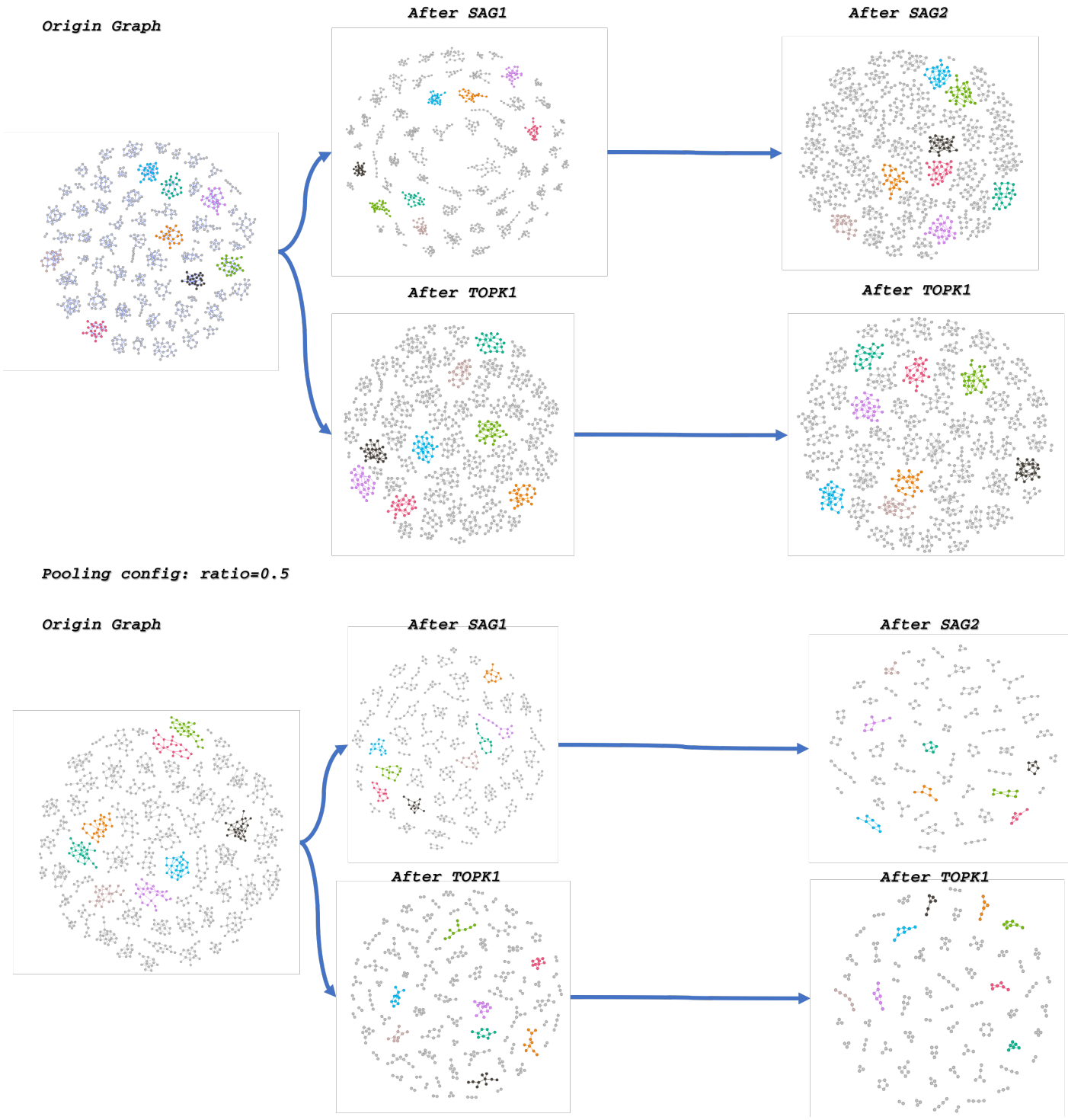

Dataset: TUDataset TRIANGLES
Pooling config: min_score=0.001
Origin Graph
After SAG1
After SAG2
After TOPK1
After TOPK1
Pooling config: ratio=0.5
Origin Graph
After SAG1
After SAG2
After TOPK1
After TOPK1

Figure 2:

In embedding domain: **Maintenance of group structure and similarity**

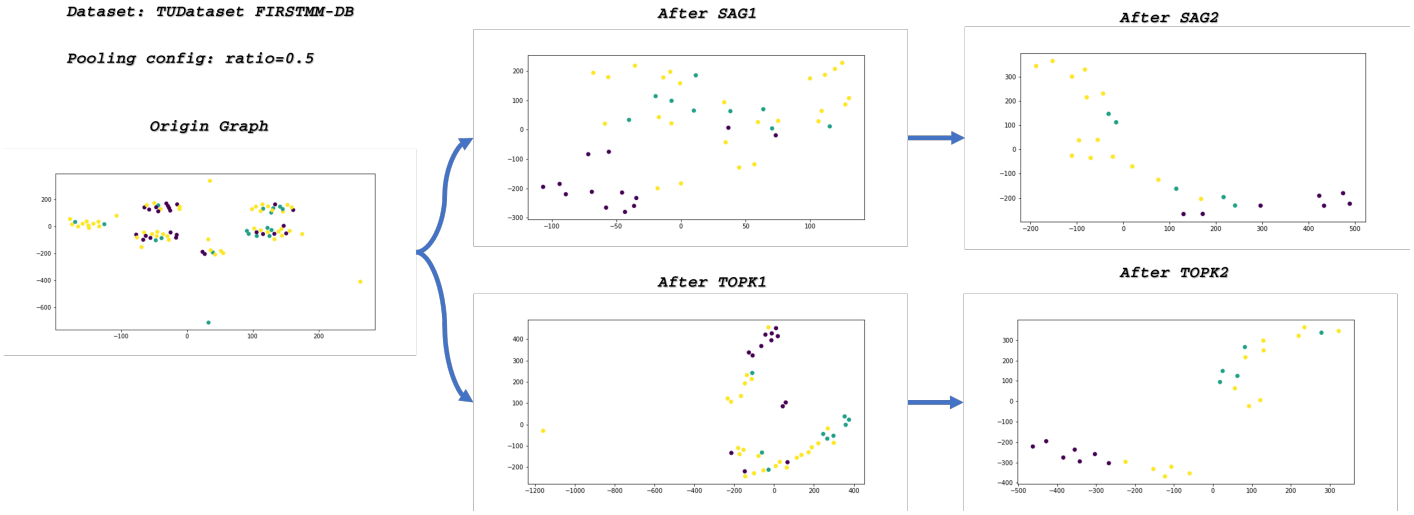


Figure 3:

Details can be found at Github link

2.5 Hierarchically Aggregated computation Graphs (HAGs)

2.5.1 Paper reading and Key idea

Represent common neighbors across different nodes using aggregation hierarchies, which eliminates redundant computation and unnecessary data transfers in both GNN training and inference.

Problems:

- Maybe Not Optimal, But No Better Idea
- Did not release code or details the heap maintenance method
- More Discussion:
  1. Understanding of the model:

     Details can be found at Github link
  2. Detailed description about max redundancy computation:

     Details can be found at Github link

2.5.2 Reimplementation

Release HAG model Version 0.0

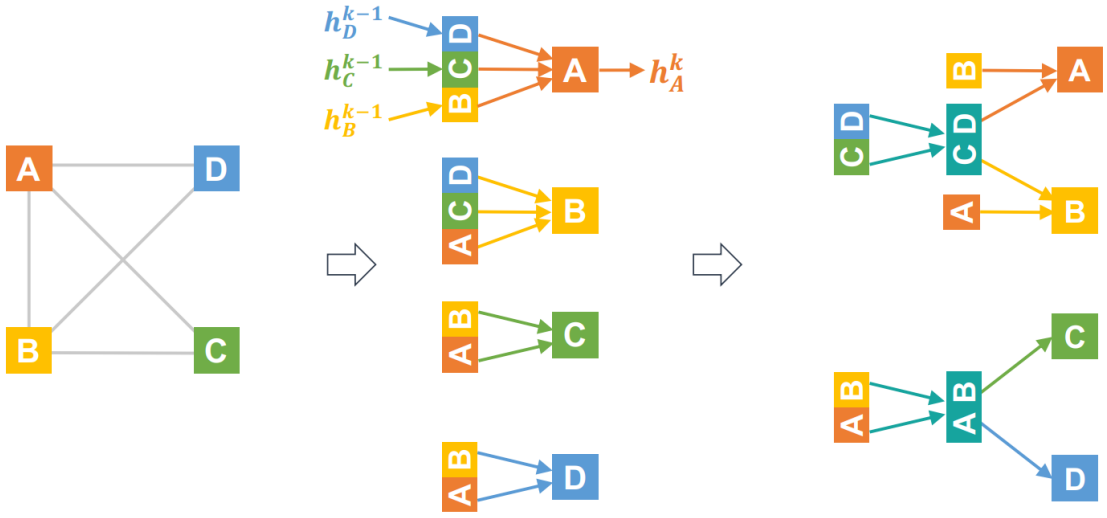
More details can be found at Tutorial and Github build

3 Motion-Vector based video Object Detection

3.1 Tasks

3.1.1 Proof of mathmatical principal of DFF

We try to proof the existence of a linear transformation that maps the feature map of key frame to the feature map of non-key frames based on the motion information:



(a) Input graph (b) GNN-graph (c) HAG

Figure 1: Comparison between a GNN-graph and an equivalent HAG. (a) Input graph; (b) 1-layer GNN computation graph (GNN-graph); (c) HAG that avoids redundant computation. The GNN-graph computes new activations $h_v^{(k)}$ by aggregating the previous-layer activations of v's neighbors. Because nodes in the input graph share common neighbors, the GNN-graph performs redundant computation (e.g., both $\{A, B\}$ and $\{C, D\}$ are aggregated twice). (c) By identifying common computational patterns, the HAG avoids repeated computation.

Figure 4:

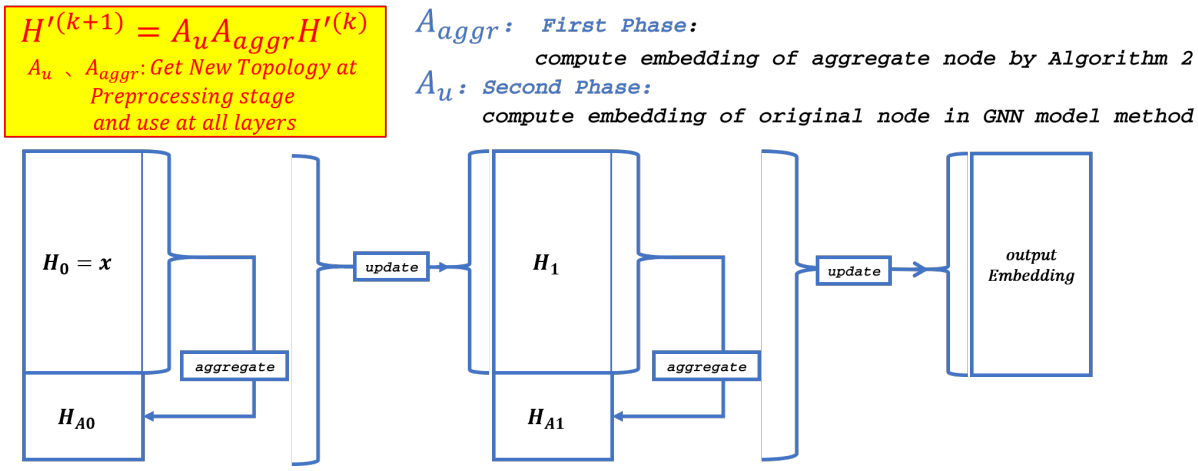


Figure 5:

and topology.bb

KEY POINT: computation graph (directed graph) is different from topology graph (directed or undirected graph) and our object is to eliminate redundancy in computation graph

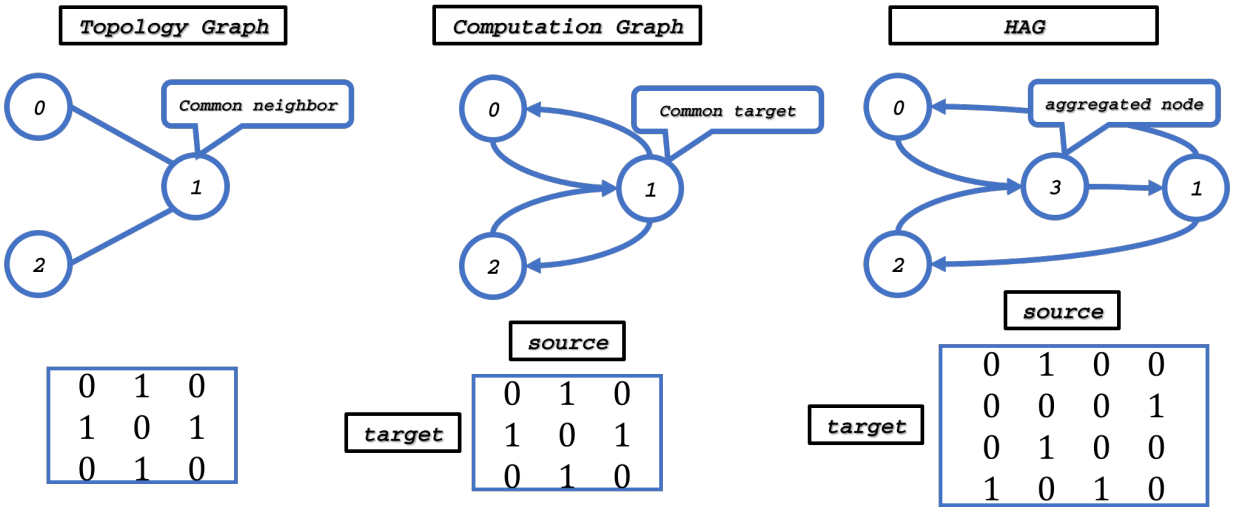


Figure 6:

Given a convolution operation $\mathbb{C}$, $\forall$ frame $\mathcal{A}$ and $\mathcal{B}$, as well as the corresponding feature maps $\mathcal{A}'$ and $\mathcal{B}'$, $\exists$ a linear transformation $\mathcal{T} = \mathcal{C}^{-1} \cdot \mathcal{M}_{\mathcal{A}\to\mathcal{B}} \cdot \mathcal{C}$, such that $\mathcal{B}' = \mathcal{A}' \cdot \mathcal{T} + \delta'$, where $\delta' = \delta C$, where $\mathcal{M}_{\mathcal{A}\to\mathcal{B}}$ and δ are motion and error information extracted from motion vector and residual map respectively.

And the error term of residual map will not explode after a sequence of convolution operations:

Given a convolution operation $\mathbb{C}$ with unit normality and an error information $\delta \sim \mathcal{N}(0, \sigma^2)$, the error information δ' after convolution operation enjoys convolution-invariance, *i.e.*, $\delta' = \delta C \sim \mathcal{N}(0, \sigma^2)$.

Details can be found at Github link

3.1.2 Motion Vector Feature Flow Version3

Idea

Rather than just scale the motion vector by 1x1 Convolutional Layer, we try to build a more complicated MV_Net try to improve the quality of motion vector used at feature map level. MV_Net is piced as following:

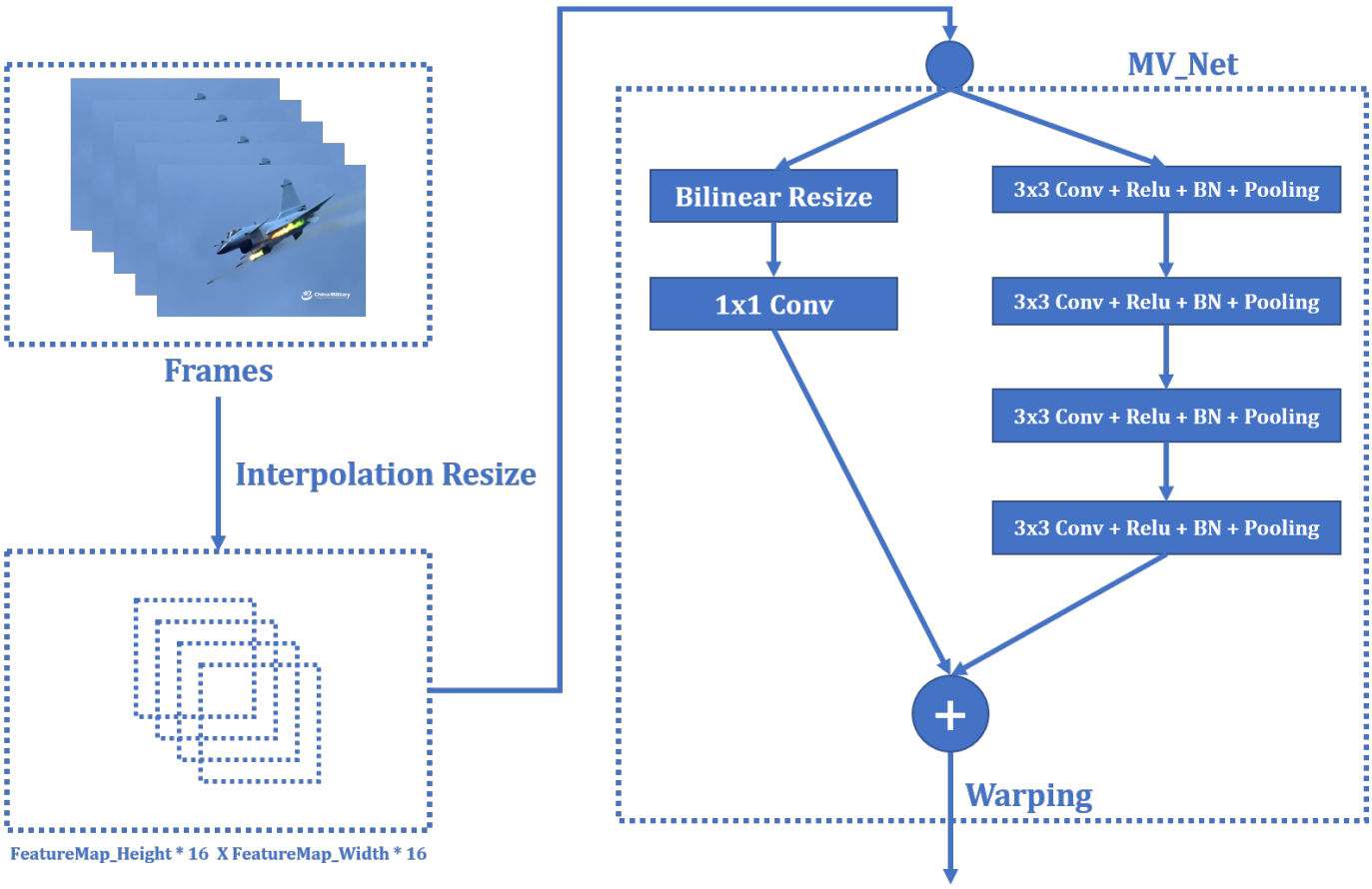


Figure 7:

Result and Discussion

We get MAP@5 = 0.6225 with above MV_Net architecture, some points we discussed in the design:

- MVFF-Object-Detection task is sensitive to the information loss in integer-times scale and width-height-same-ratio scale of movtion vector in pooling process, so we need firstly use interpolation (non-integer-times) scale to scale the movtion vector to a integer-times of
  feature map shape (16*feat-map-width, 16*feat-map-height)

  Details can be found at Github link

Code find at MVFF-Version3

File organization same as Deep Feature Flow for Video Recognition

3.1.3 Motion Vector Feature Flow Version4

Idea

Use DMC-Net like structure to fine-tune the motion vector, try to gain more motion information form residual data under optical flow guidence.

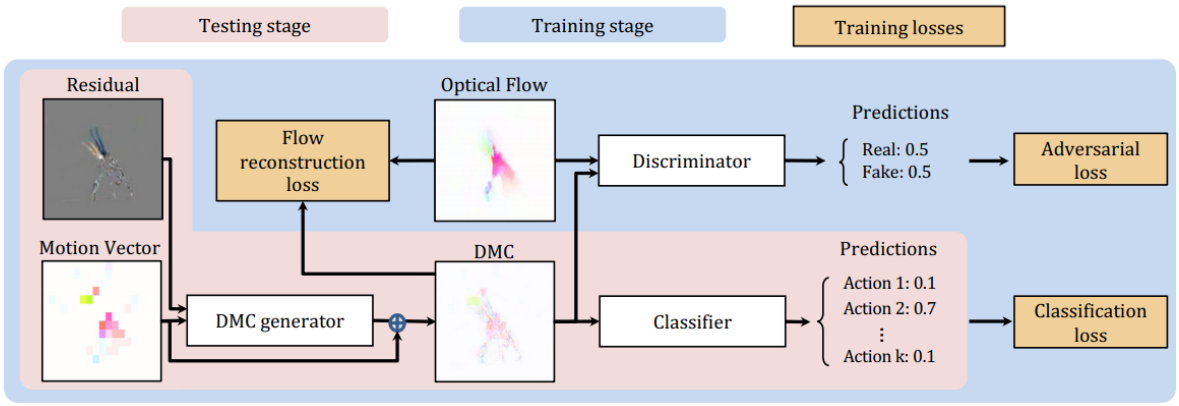


Figure 3: The framework of our Discriminative Motion Cue Network (DMC-Net). Given the stacked residual and motion vector as input, the DMC generator reduces noise in the motion vector and captures more fine motion details, outputting a more discriminative motion cue representation which is used by a small classification network to classify actions. In the training stage, we train the DMC generator and the action classifier jointly using three losses. In the test stage, only the modules highlighted in pink are used.

Layer	Input size	Output size	Filter config
conv0	5, 224, 224	8, 224, 224	8, 3x3, 1, 1
conv1	13, 224, 224	8, 224, 224	8, 3x3, 1, 1
conv2	21, 224, 224	6, 224, 224	6, 3x3, 1, 1
conv3	27, 224, 224	4, 224, 224	4, 3x3, 1, 1
conv4	31, 224, 224	2, 224, 224	2, 3x3, 1, 1
conv5	33, 224, 224	2, 224, 224	2, 3x3, 1, 1

Table 2: The architecture of our Discriminative Motion Cue (DMC) generator network which takes stacked motion vector and residual as input. Input/output size follows the format of #channels, height, width. Filter configuration follows the format of #filters, kernel size, stride, padding.

DMC-Net details can be found at Github Link

Result and Discussion

Result of MVFF-Version4-without optical flow guidence: MAP@5 = 0.5091.

Maybe we need extract optical flow from the dataset first.

Code find at MVFF-Version4

3.1.4 Motion Vector Output Flow Step-Performance Curve

We tried different steps of MVOF to approximate the result of DFF and accelerate it, trying to analyse motion vector propagation at output level.

STEP	2	3	4	6	12
MAP@5	0.6982	0.6921	0.6898	0.6797	0.648

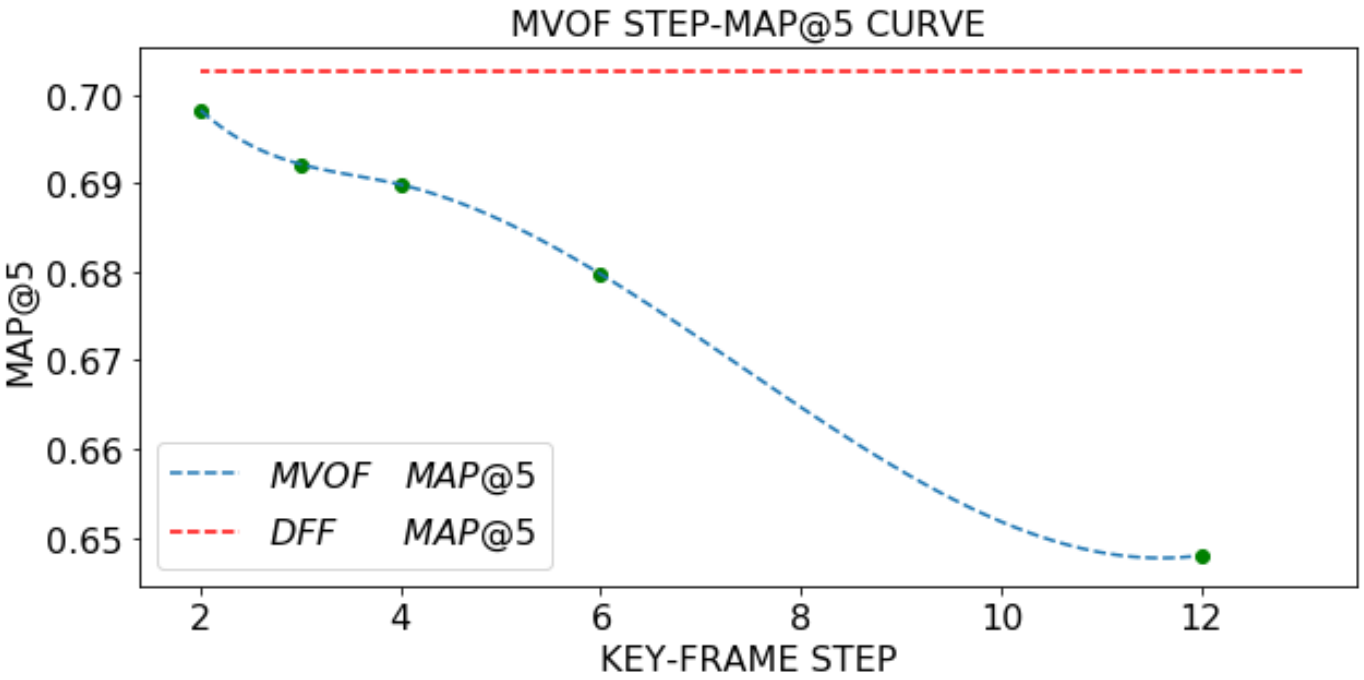


Figure 8:

4 Quantum Computing Learning

4.1 First stage: From bits to qubits: Basical Concepts and Algorithm of Quantum Computing

4.1.1 What Is a QPU?

- QPU (Quantum Processing Unit) is a co-processor
- QPU has ability to dramatically extend the kinds of problems that are tractable within computing.
- The CPU issues the QPU co-processor commands only to initiate tasks suited to its capabilities.

4.1.2 Native QPU Instructions

- Conventional high-level languages are commonly used to control lower-level QPU instructions.
- Essential QPU instruction set

4.1.3 Simulator Limitations

- One measure of the power of a QPU is the number of qubits it has available to operate on.

- Each qubit added to simulation will double the memory required to run the simulation, cutting its speed in half.

4.1.4 QPU Versus GPU:

Some Common Characteristics What it's like to program a QPU:

- It is very rare that a program will run entirely on a QPU. Usually, a program runnning on a CPU will issue QPU instructions, and later retrieve the results.
- Some tasks are very suited to the QPU, and others are not.
- The QPU runs on a separate clock from the CPU, and usually has its own dedicated hardware interfaces to external devices (such as optical outputs).
- A typical QPU has its own special RAM, which the CPU cannot efficiently access.
- A simple QPU will be one chip accessed by a laptop, or even perhaps eventually an area within another chip. A more advanced QPU is a large and expensive addon, and always requires special cooling.
- When a computation is done, a projection of the result is returned to the CPU, and most of the QPU's internal working data is discarded.
- QPU debugging can be very tricky, requiring special tools and techniques, Stepping through a program can be difficult, and often the best approcah is to make changes to the program and observe their effect on the output.
- Optimizations that speed up one QPU may slow down another.

4.2 ~~Second stage: Great idea evolution and Important Works~~

4.3 ~~Third stage: On-going Front Problem and Research~~

4.4 ~~Fourth stage: Research directions~~