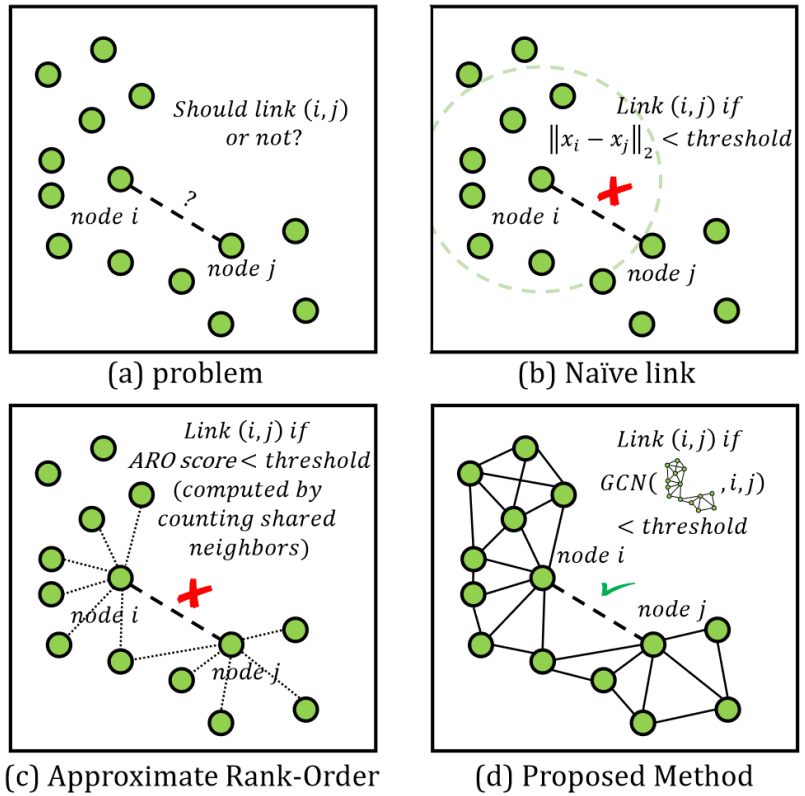
Abstract.

Task :
얼굴들 잘 클러스터링하기 -> '링크 예측 문제' 로 치환. : 동일한 사람의 얼굴 사이에는 링크 존재한다고 가정함.

Key idea :
Feature space 상에 instance 주변의 local context가 그 instance 와 이웃들 간의 링크 정보를 많이 포함하고 있다.

GCN(Graph Convolutional Network).
Input : 각 instance 주변의 sub-graph : local context
Output : sub-graph 내의 pair 간의 링크 가능성을 infer함.

Instruction.



(a) problem (b) Naïve link

(c) Approximate Rank-Order (d) Proposed Method

Figure 1. High-level idea of our method. (a): This paper aims to estimate whether two nodes should be linked. (b-d): Comparison of three linkage estimation methods. (b): Directly thresholding the l_2 distance, *without considering context*. (c): Using a heuristic mthod for linkage estimation *based on context*. (d): Our method, *i.e.*, learning the linkage likelihood with a parametric model, which is *context-based*.

-기존 클러스터링 방법의 한계 : 데이터 분포에 대한 비현실적인 가정을 함.
K-means : 클러스터가 볼록 형태여야함.
Spectral clustering : 각 클러스터가 동일한 수의 인스턴스를 가져야 함.
DBSCAN : 각 클러스터가 동일한 밀도를 가져야 함.
반면, 링크 기반 클러스터링 방법들은 데이터 분포에 대한 가정을 하지 않음.

-Figure1(a) :
링크 기반 클러스터링 방법
= 두 노드가 연결되어야 하는지 예측
= 두 클러스터가 동일한 정체성을 가지는지 예측

-Figure1(b) :
두 노드 간 l2 거리가 특정 임계값 이하이면 연결. (이는 클러스터마다 밀도가 크게 다르기 때문에 좋은 해결책이 아님)

-Figure1(c) :
더 정교한 메트릭으로 링크 가능성 예측.

-Figure1(d) : Ours :
휴리스틱 메트릭을 사용해 링크 가능성을 예측하기 보다는, 두 노드가 연결되어야 하는지 예측하는 방법을 '학습'하자.
노드와 이웃 간의 링크 가능성은 해당 노드의 'context'로부터 유추.
GCN 기반한 학습 가능한 클러스터링 방법.

-Ours :
1. 클러스터링 문제를 링크 예측 문제로 치환. (링크는 라벨이 동일한 두 노드 간에만 존재)
2. Instance 와 근처 이웃들 간의 링크만 예측.
따라서, 각 instance(pivot) 주변에 IPS(Instance Pivot Subgraph)를 만듬.
IPS는 local context를 묘사함.
각 노드는 pivot-neighbor 쌍을 모델링함.
GCN : IPS로부터 pivot-neighbor 쌍이 어떤식으로 연결되어야 할지 학습함.
3. GCN의 output : 링크 가능성 집합.
우리는 연결된 노드를 전이적으로 병합하여 클러스터를 얻음.

Related work.

Face clustering.

Link prediction.

Graph convolutional network (GCN).
Ours :
1. spatial-based GCN : 그래프 노드와 그 이웃에 대해 수동으로 정의된 컨볼루션을 직접 수행.
2. Inductive setting : 테스트 시 모델이 다른 그래프에서 추론해야함.

Proposed Approach.

3.1. Overview.

Problem definition.
얼굴 이미지 features : $X = [\boldsymbol{x_1}, ..., \boldsymbol{x_N}]^T \in \mathbb{R}^{N \times D}$
N : 이미지 개수
D : feature의 차원

얼굴 클러스터링의 목표 : pseudo label y_i 부여하여 같은 pseudo label 할당된 instance들끼리 클러스터 형성. $i \in \{1, 2, ..., N\}$
➔ Ours : link-based clustering : instance pair 간에 링크 가능성을 예측. 링크로 연결된 instance들끼리 클러스터 형성.

Motivation.
모든 잠재적인 pair을 다 고려하지 않고, instance와 k개의 근접 이웃들 간의 링크 가능성을 계산하는 것만으로도 충분하다.

k	5	10	20	40	80	160
F-measure	0.874	0.911	0.928	0.946	0.959	0.970
NMI	0.960	0.969	0.975	0.981	0.986	0.990

Table 1. Upper bound of face clustering accuracy on the IJB-B-512 dataset. We use two metrics for evaluation, F-measure and NMI (see Section 4.1). The upper bound is obtained by connecting each instance with its k nearest neighbors *if the neighbor is of the same identity with this instance.* We find that the upper bound is reasonably high, indicating that kNN method could be effective while, most importantly, being efficient.

Pipeline.
Instance와 kNNs 사이에 링크 예측.

IPS(Instance Pivot Subgraphs)
: pivot instance p가 중심에 있는 subgraph.
: p의 kNNs과 p의 h-hop까지 고차 이웃들을 포함하는 노드들로 구성.

중요한점은, 모든 노드들에서 pivot p의 feature를 빼서, 각 노드의 feature가 pivot-neighbor pair의 링크관계를 인코딩함.

1. 모든 instance를 pivot으로 사용하여 IPS를 구성함. (-> 3.2)
2. GCD의 인풋 = IPS, 아웃풋 = 모든 노드에 대한 점수 = pivot-neighbor pair 간에 링크 가능성. (-> 3.3)
3. 아웃풋 = 전체 그래프 상에서 weighted edge 집합. Weight은 링크 가능성. 링크 가능성에 따라 linked instance들을 병합하여 클러스터 형성. (-> 3.4)

3.2. Construction of Instance Pivot Subgraph

두 얼굴 이미지 (노드) 간에 링크 가능성을 그래프 내의 context 기반해서 예측. IPS를 context로 구성.

IPS 생성 과정 : 3단계 : Figure 2
1. IPS의 모든 노드를 위치.
2. Pivot feature를 뺌으로서 node feature를 정규화.
3. 노드 간에 엣지 추가.

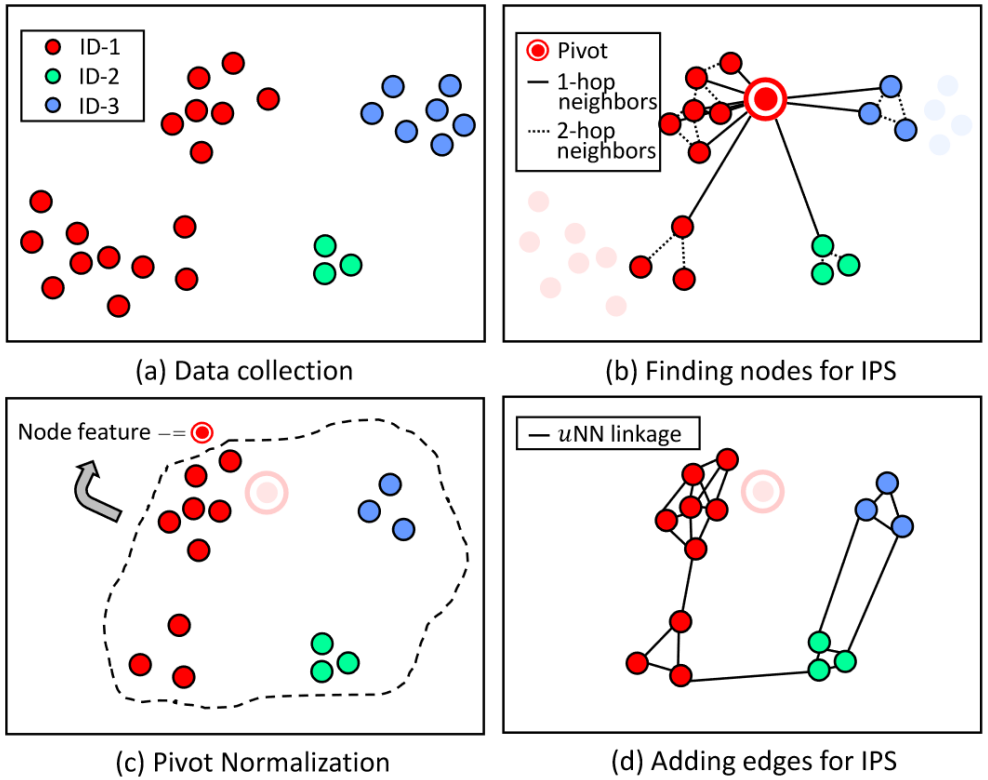


Figure 2. Construction of Instance Pivot Subgraph (IPS). (a) The collection of face representations. (b) We use each instance p as a pivot, and find its neighbors up to h-hop as the nodes of IPS. (c) These node features are normalized by subtracting the feature of the pivot. (d) For a node in IPS, we find its uNNs from the entire collection. We add an edge between a node and its uNNs if the neighbor is also a node of IPS. In this figure, we set $h = 2, k_1 = 10, k_2 = 2$ and $u = 3$, where k_1 is the number of 1-hop neighbors, and k_2 is the number of of 2-hop neighbors. Note that an IPS based on pivot p does not contain p. The IPS for pivot p is used to predict the linkage between p and every node in IPS.

Step 1: Node discovery.
Pivot p에 대하여 h-hop까지의 이웃들을 IPS의 노드로 사용함.
k_i : i-th hop에서 근처 이웃 개수 $i = 1, 2, ..., h$

p : pivot
V_p: node set (pivot p 불포함)
$G_p(V_p, E_p)$: IPS

고차 이웃들은 pivot과 이웃 간의 context의 로컬 구조에 대한 보조적 정보를 제공함.

Step 2: Node feature normalization.
p : pivot instance -> $\boldsymbol{x_p}$: pivot feature
V_p : node set -> $\{\boldsymbol{x_q} | q \in V_p\}$: node features

Pivot 정보를 IPS의 node features에 인코딩하기 위해, node features에 pivot feature을 빼는 방식으로 정규화 시킴.
$\boldsymbol{\mathcal{F}_p}$: 정규화된 node features
$\boldsymbol{\mathcal{F}_p} = [\ldots, \boldsymbol{x_q} - \boldsymbol{x_p}, \ldots]^T$, for all $q \in V_p$ $\quad\quad$ $\boldsymbol{\mathcal{F}_p} \in \mathbb{R}^{|V_p| \times D}$

IPS 내 node feature : pivot p의 feature와 neighbor q의 feature 사이의 residual vector.

Step 3: Adding edges among nodes.
노드 $q \in V_p$ 에 대하여 전체 instance들 중 제일 가까운 u개의 이웃을 찾음.
uNNs 중 노드 r이 V_p에 등장하면, 엣지 (q, r) 을 엣지 집합 E_p에 추가함.
IPS 구조는 인접행렬 $\boldsymbol{A_p} \in \mathbb{R}^{|V_p| \times |V_p|}$, node feature 행렬 $\boldsymbol{\mathcal{F}_p}$로 표현함.
(앞으로 아래첨자 p는 생략)

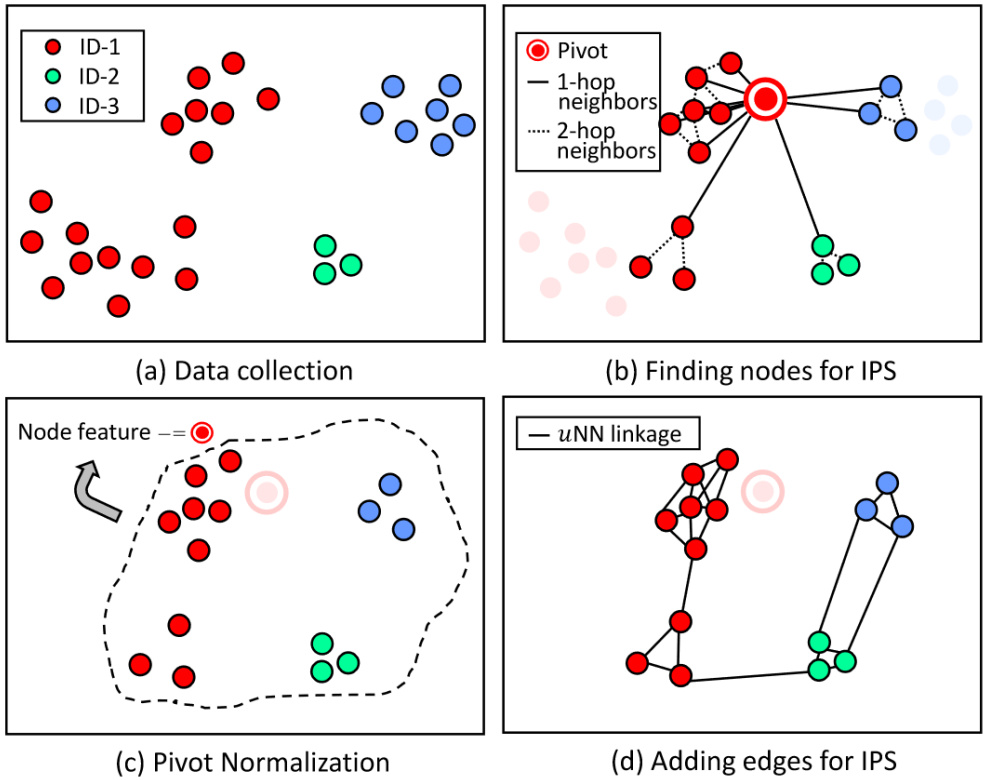


Figure 2. Construction of Instance Pivot Subgraph (IPS). (a) The collection of face representations. (b) We use each instance p as a pivot, and find its neighbors up to h-hop as the nodes of IPS. (c) These node features are normalized by subtracting the feature of the pivot. (d) For a node in IPS, we find its uNNs from the entire collection. We add an edge between a node and its uNNs if the neighbor is also a node of IPS. In this figure, we set $h = 2, k_1 = 10, k_2 = 2$ and $u = 3$, where k_1 is the number of 1-hop neighbors, and k_2 is the number of of 2-hop neighbors. Note that an IPS based on pivot p does not contain p. The IPS for pivot p is used to predict the linkage between p and every node in IPS.

3.3. Graph Convolutions on IPS

IPS에 포함된 context (노드 간 엣지) 는 노드가 positive인지 negative인지 (pivot 과 link할지 안할지) 결정하는데 굉장히 중요함.
GCN의 인풋 = node feature matrix $\boldsymbol{X}$ 와 adjacency matrix $\boldsymbol{A}$
GCN의 아웃풋 = 변형된 node feature matrix $\boldsymbol{Y}$

첫번째 layer에서 input node feature matrix = original node feature matrix. $\boldsymbol{X} = \boldsymbol{\mathcal{F}}$
Graph convolution layer : $\boldsymbol{Y} = \sigma([\boldsymbol{X} \| \boldsymbol{G}\boldsymbol{X}]\boldsymbol{W})$
$\boldsymbol{X} \in \mathbb{R}^{N \times d_{in}}$
$\boldsymbol{Y} \in \mathbb{R}^{N \times d_{out}}$
N : 노드 개수
d_{in}, d_{out} : input/output node feature의 차원

$\boldsymbol{G} = \boldsymbol{g}(\boldsymbol{X}, \boldsymbol{A})$: aggregation matrix. $N \times N$ 크기. 각 행의 합은 1.
$\|$: feature 차원에서 행렬 연결.
$\boldsymbol{W}$: graph convolution layer의 학습 가능한 weight matrix. $2d_{in} \times d_{out}$ 크기.
$\sigma(\cdot)$: 비선형 활성 함수.

Graph convolution 연산 : 2 step.
1. $\boldsymbol{G}\boldsymbol{X}$: 노드 이웃들의 기본 정보 집계
$\boldsymbol{X} \| \boldsymbol{G}\boldsymbol{X}$: input node feature $\boldsymbol{X}$ 와 aggregated information $\boldsymbol{G}\boldsymbol{X}$ 을 feature 차원 따라 concat 함.
2. Concat 된 feature들은 linear filters에 의해 변형됨. (linear filter에서 parameter $\boldsymbol{W}$ 가 학습됨)

집계 연산에 대한 3가지 전략 $\boldsymbol{g}(\cdot)$
Mean aggregation
Weighted aggregation
Attention aggregation

Mean aggregation.
$\boldsymbol{G} = \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{\Lambda}^{-\frac{1}{2}}$: aggregation matrix
$\boldsymbol{A}$: adjacency matrix
$\boldsymbol{\Lambda}$: diagonal matrix $\boldsymbol{\Lambda}_{ii} = \sum_j \boldsymbol{A}_{ij}$
Mean aggregation : 이웃들 간에 average pooling

Weighted aggregation.
$\boldsymbol{A}$ 에서 0이 아닌 원소들을 해당 cosine similarity로 대체하고, 각 행마다 0이 아닌 원소들을 softmax를 사용하여 정규화함.
Weighted aggregation : 이웃들 간에 average pooling

Attention aggregation.
이웃들의 aggregation weights 을 학습.
$\boldsymbol{G}$ 안의 원소들은 pivot-neighbor nodes pair의 feature들을 2-layer MLP에 인풋으로 넣어서 생성.
Attention aggregation : 이웃들 간에 weighted average pooling (weights 은 자동으로 학습됨)

이 논문에서 사용한 GCN :
ReLU로 활성화된 4개의 graph convolution layers로 구성.
Softmax로 활성화 후 CE loss 사용.
1-hop neighbors nodes의 gradient만 backpropagate. (왜냐하면 우리는 pivot과 1-hop neighbors 간의 링크만 고려)
테스트 시에도, 1-hop nodes 에 대해서만 노드 분류 수행.

Graph convolution 예시)
2D input node feature
2 graph convolution layers (각 layer의 output dimension : d_1 = 2, d_2 = 2)

Figure 4 : 각 layer의 output embedding이 training iteration에 따라 어떻게 변하는지.

각 graph convolution layer 이후, positive nodes(빨강)는 더 가까이 그룹화 되고 negative nodes(초록, 파랑)는 다른 그룹을 형성함.
(이웃들의 메시지가 aggregation 단계에서 노드들에 전달되기 때문에, 그리고 이웃들의 메시지가 링크된 노드들을 당기는 임베딩에 부드러운 역할을 함)
한편 supervision signal은 positive nodes와 negative nodes는 서로 밀어냄.
마침내, 시스템은 positive groups와 negative groups은 서로 멀어지고, 같은 카테고리 내에 있는 nodes는 서로 가까워지는 균형점에 도달함.

3.4. Link Merging

모든 인스턴스를 순회하며 각 인스턴스를 pivot으로 하여 IPS를 구성.
해당 인스턴스와 pivot 간의 링크 가능성(노드 분류기가 출력하는 소프트맥스 확률)을 예측.
링크 가능성에 의해 가중치가 부여된 엣지 집합.
가중치가 특정 임계값 이하인 모든 엣지를 잘라내고, Figure 3에 나타난 것처럼 너비 우선 탐색(BFS)을 사용하여 pseudo label 을 propagate.
각 반복에서 알고리즘은 임계값 이하의 엣지를 잘라내고, 사전 정의된 최대 크기보다 큰 connected clusters을 유지하여 queue에 넣음(다음 반복에서 처리 예정).
다음 반복에서는 엣지를 자르는 임계값 증가.
이 과정을 queue가 빌때까지 반복(모든 인스턴스가 pseudo label로 표시될 때까지).

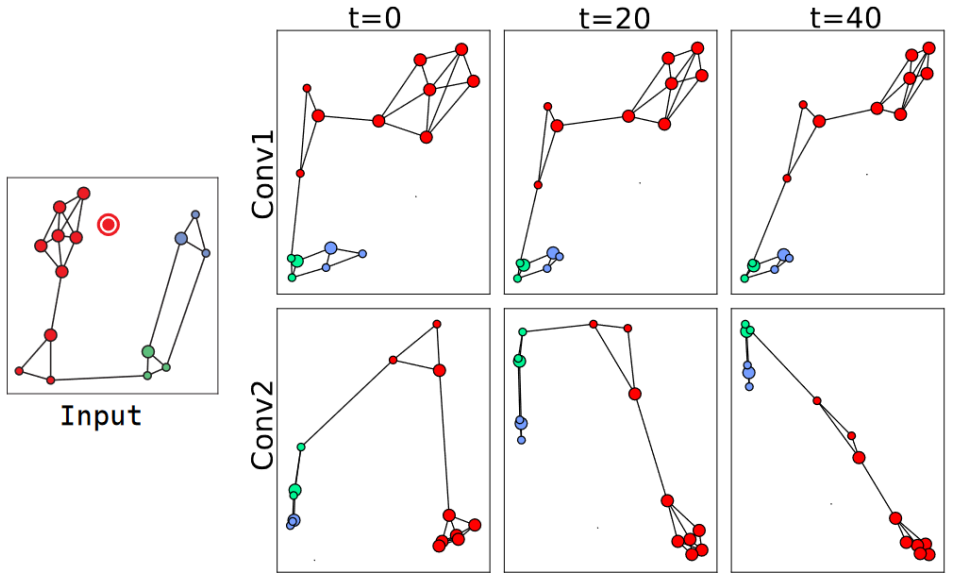


Figure 4. A toy example to illustrate the working mechanism of graph convolutions. Different colors refer to different IDs. The pivot is circled. Training gradients are bac-kpropagated for 1-hop neighbors (larger nodes) but not for higher-order neighbors (smaller nodes). We observe that, after each graph convolution, positive and negative groups become farther from each other and nodes in the same category become closer .

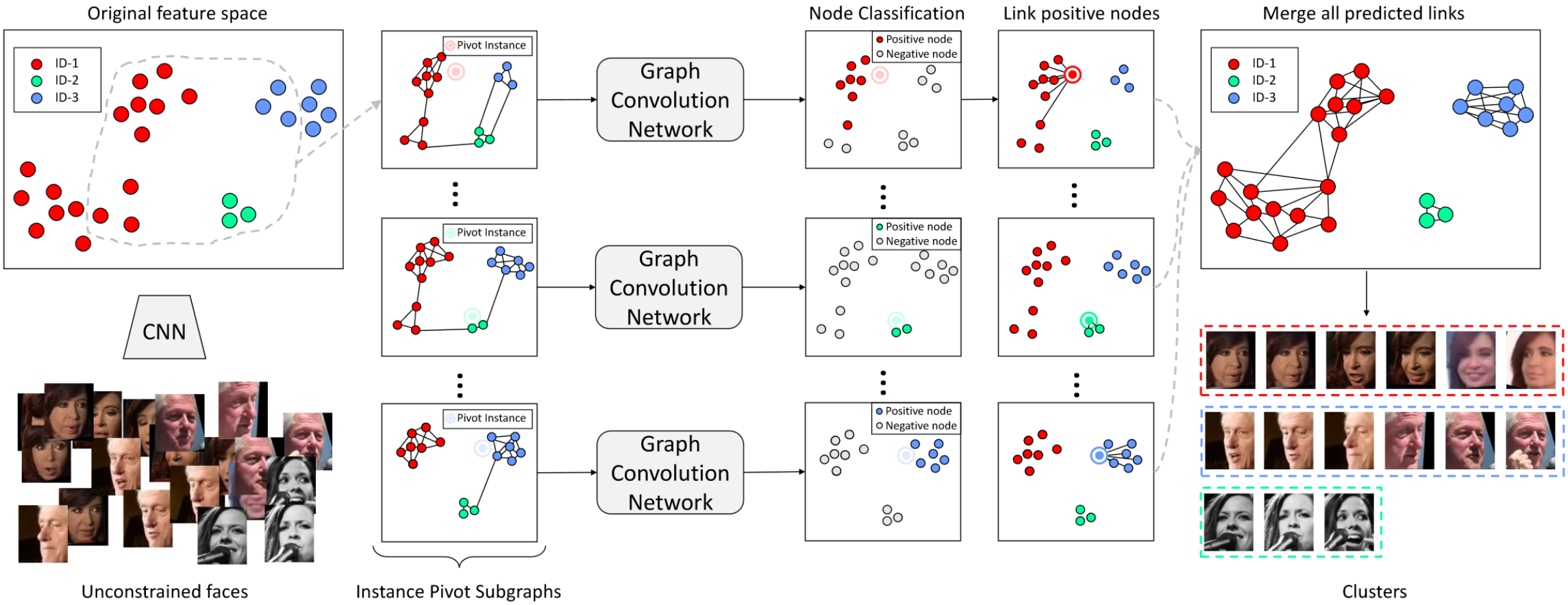


Figure 3. Framework of our method. We use every instance as a pivot, and construct an Instance Pivot Subgraph (IPS) for it. Each node in IPS models the linkage (similarity) between the pivot and the corresponding neighbor. We apply graph convolutions on IPS and classify nodes in IPS into either positive or negative. If a node is classified as positive, the corresponding neighbor should be linked to the pivot. After determining all the linkages, we transitively merge linked instances to obtain the final clusters.