# AI3607 Deep Learning and Its Application
# Project: **L**inear **A**ssignment **P**roblem Based **G**raph **M**atching

蒋伟, F2003801, 520030910149

(Dated: January 8, 2023)

## 1 Introduction

Graph matching problem is a fundamental problem in deep learning. This combinational optimization problem aims to find node correspondences (and then naturally edge correspondences) among graphs, which frequently occur as the data structure in recommendation system, knowledge graph, and computer vision. Traditional methods integrate both node-wise similarity and edge-wise similarity to optimize node correspondences, leading to a quadratic assginment programming (QAP) problem which is hard to solve.

In this project, we will focus on some recently-proposed linear assignment problem (LAP) based graph matching algorithms, which only consider similarity at node level. To make full use of given information, these algorithms utilize powerful deep neural networks to fuse the graph structure and edge features all into node features, and then solve the graph matching problem as a linear assignment problem. With Sinkhorn [10] algorithm, the solving process is even differentiable, which makes it possible to train the whole graph matching pipeline end-to-end.

In practice, we will implement a universal LAPGM model trainer compatible with PIA-GM, PCA-GM, IPCA-GM [13], and CIE [14] based on Jittor [7] and Pygmtools [11].

The whole project is available at https://github.com/Ailon-Island/LAPGM.

## 2 Methods

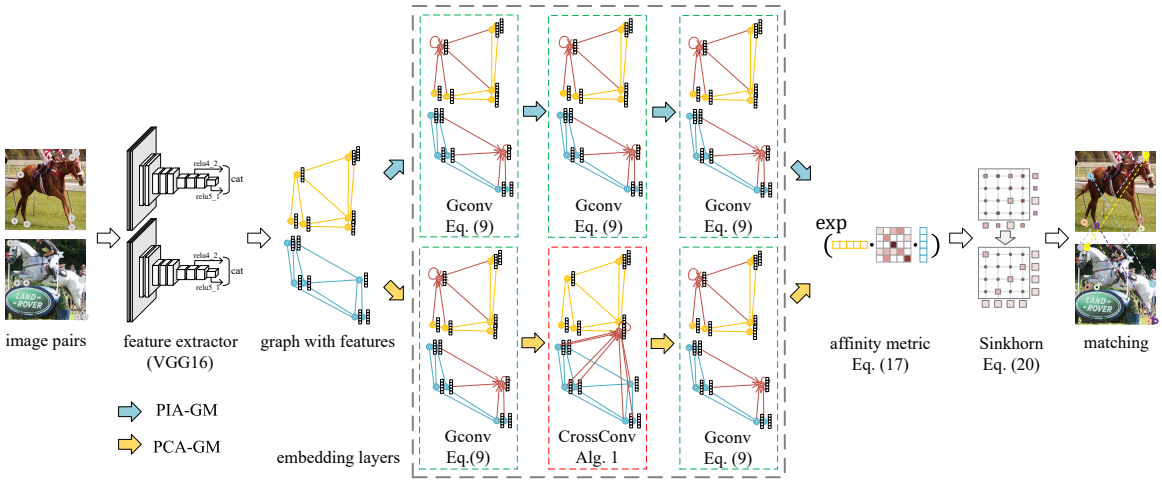
### 2.1 Image to Graph

In the project, we follow [13] and [14] to work in the setting of graph matching on image keypoints, where we need to first convert images with keypoints into featured graphs to be matched. To construct a graph, we perform Delaunay triangulation [5] on the keypoints. As for node features, we use keypoint descriptors $\mathbf{h}_{si}^{(0)}$ interpolated from CNN (VGG16 [9] in implementation) feature map. Furthermore, there are various choices for edge features $\mathbf{e}_{sij}^{(0)}$ if necessary. For simplicity, we use the edge length as the edge feature. Note that it is also possible to use the edge angle or the descriptors from the connected nodes.

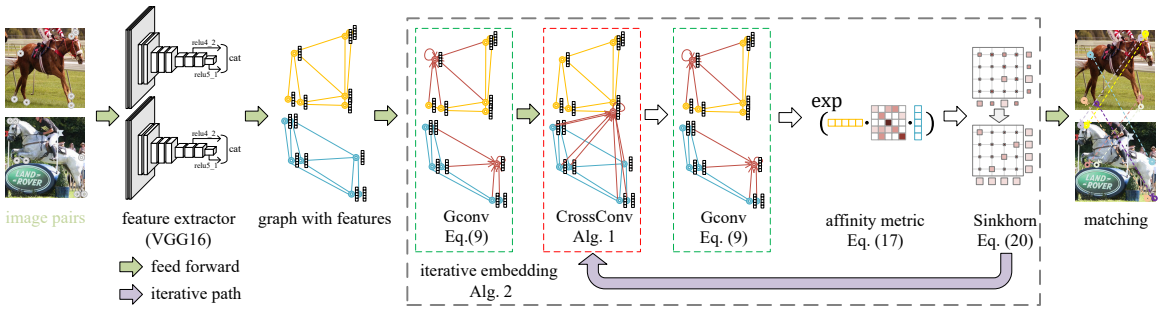
### 2.2 PIA-GM

**P**ermutation loss and **i**ntra-graph **a**ffinity based **g**raph **m**atching (PIA-GM) proposed in [13] is the first method to adopt deep graph networks to compress a graph into node feature vectors. And they are then able to for the first time introduce the Sinkhorn [10] based permutation loss to solve the graph matching problem.

To be specific, PIA-GM first uses a graph embedding network to compress features around into each node. Then, the node feature vectors $\{\mathbf{h}_{si}\}$ are fed into an affinity network to obtain a node-wise affinity matrix $\mathbf{M}$.




(a) PIA-GM and PCA-GM.




(b) IPCA-GM.

Figure 1: Overview of methods proposed in [13]: PIA-GM, PCA-GM, and IPCA-GM. Pictures adopted from [13].

Next, Sinkhorn layer is used to convert the affinity matrix into a soft assignment in the form of a doubly-stochastic matrix $\mathbf{S}$. Finally, the permutation loss is applied to optimize the node correspondences. For final binary permutation matrix $\mathbf{X}$, we can perform Hungarian [8] algorithm on $\mathbf{S}$ as a final discretization step.

**Intra-graph node embedding.** The intra-graph node embedding procedure fuses structural information into nodes so as to obtain $\{\mathbf{h}_{si}\}$ from $\left\{\mathbf{h}_{si}^{(0)}\right\}$. This is also called graph convolution (GConv). We apply GConv several times for gradually global information aggregation. One GConv step is defined as follows:

$$\mathbf{m}_{si}^{(k)} = \frac{1}{|(i,j) \in \mathcal{E}_s|} \sum_{j:(i,j)\in\mathcal{E}_s} f_{\text{msg}}\left(\mathbf{h}_{si}^{(k-1)}\right) \tag{1}$$

$$\mathbf{n}_{si}^{(k)} = f_{\text{node}}\left(\mathbf{h}_{si}^{(k-1)}\right) \tag{2}$$

$$\mathbf{h}_{si}^{(k)} = f_{\text{update}}\left(\mathbf{m}_{si}^{(k)}, \mathbf{n}_{si}^{(k)}\right) \tag{3}$$

where $\mathcal{E}_s$ is the set of edges in $s$-th graph, $f_{\text{msg}}$, $f_{\text{node}}$, and $f_{\text{update}}$ are the message passing function, the self-passing function, and the update function, respectively. During the steps, $\mathbf{m}_{si}$ aggregates messages from neighbors of node $i$ and $\mathbf{n}_{si}$ processes the current node feature, then they are combined to update the node feature $\mathbf{h}_{si}$.

**Affinity network.** With all node features at hand, we need an affinity matrix $\mathbf{M}$ to describe node-to-node similarity between two graphs:

$$\mathbf{M}_{ij} = f_{\text{aff}}\left(\mathbf{h}_{1i}, \mathbf{h}_{2j}\right), \quad i \in \mathcal{V}_1, j \in \mathcal{V}_2$$

where $f_{\text{aff}}$ is the affinity score function, which is a weighted bi-linear function followed by an exponential function

$$f_{\text{aff}}\left(\mathbf{h}_{1i}, \mathbf{h}_{2j}\right) = \exp\left(\frac{\mathbf{h}_{1j}^{\top}\mathbf{K}\mathbf{h}_{2j}}{\tau}\right)$$

where $\mathbf{K}$ is the learnable weight, $\tau$ is the hyperparameter of temperature. for $\tau > 0$, with $\tau \to 0^+$, $f_{\text{aff}}$ becomes more discriminative, while the chance of explosive gradient climbs.

**Sinkhorn layer.** Sinkhorn [10] method is used as a relaxed approximation of Hungarian algorithm [8] for LAPs. It takes in any non-negative square matrix and outputs a doubly-stochastic matrix as prediction. The Sinkhorn operator is

$$\mathbf{M}'^{(k)} = \mathbf{M}^{(k-1)} \oslash \left(\mathbf{M}^{(k-1)}\mathbf{1}\mathbf{1}^\top\right) \tag{4}$$

$$\mathbf{M}^{(k)} = \mathbf{M}'^{(k)} \oslash \left(\mathbf{1}\mathbf{1}^\top\mathbf{M}'^{(k)}\right) \tag{5}$$

$$\tag{6}$$

where $\oslash$ means element-wise division, and $\mathbf{1}$ is an all-one column vector. In other word, Sinkhorn algorithm alternately takes row-normalization and column-normalization till convergence. For two graphs with different numbers of nodes, we can pad with some isolated dummy nodes.

With the doubly-stochastic prediction, it is natural to utilize cross-entropy loss as the objective, which is called "permutation" loss in [13]:

$$\mathcal{L}_{\text{CE}} = -\sum_{i\in\mathcal{V}_1, j\in\mathcal{V}_2} \left(\mathbf{X}_{ij}^{\text{gt}} \log \mathbf{S}_{ij} + \left(1 - \mathbf{X}_{ij}^{\text{gt}}\right) \log\left(1 - \mathbf{S}_{ij}\right)\right)$$

## 2.3 PCA-GM

As is proposed in [13], **p**ermutation loss and **c**ross-graph **a**ffinity based **g**raph **m**atching (PCA-GM) is an improvement from PIA-GM. It introduces cross-graph fusion to boost cross-graph similarity discovery. For node $\mathbf{h}_{1i}$ in the first graph, the new cross-graph node embedding procedure (CrossConv) is defined as follows:

$$\mathbf{m}_{1i}^{(k)} = \sum_{j\in\mathcal{V}_2} \hat{\mathbf{S}}_{i,j} f_{\text{msg}-\text{cross}}\left(\mathbf{h}_{2j}^{(k-1)}\right) \tag{7}$$

$$\mathbf{n}_{1i}^{(k)} = f_{\text{node}-\text{cross}}\left(\mathbf{h}_{1i}^{(k-1)}\right) \tag{8}$$

$$\mathbf{h}_{1i}^{(k)} = f_{\text{update}-\text{cross}}\left(\mathbf{m}_{1i}^{(k)}, \mathbf{n}_{1i}^{(k)}\right) \tag{9}$$

where the node-to-node similarity $\hat{\mathbf{S}}$ is the doubly-stochastic matrix predicted from $\left\{\mathbf{h}_{si}^{(k-1)}\right\}$ with the affinity followed by the Sinkhorn layer.

## 2.4 IPCA-GM

Note that PCA-GM gets a pre-result $\hat{\mathbf{S}}$ for cross-graph aggregation. Then the final result $\mathbf{S}$ can be interpreted as a refinement of $\hat{\mathbf{S}}$. So, it is natural to perform this refinement iteratively, which is the main idea of **i**terative **p**ermutation loss and **c**ross-graph **a**ffinity based **g**raph **m**atching (IPCA-GM) [13]. The iterative refinement procedure is to do GConv and CrossConv alternately. Note that $\hat{\mathbf{S}}$ is calculated at the end of each CrossConv, GConv pair rather than before each CrossConv, which means the very first CrossConv is a "GConv" with different parameters.

## 2.5 CIE

In all methods above, while structural features are perserved, edge attributes are neglected. **C**hannel-**I**ndependent **E**mbedding (CIE) [14] is then proposed to introduce edge attributes to LAPGM method.

Taking edge embeddings $\{\mathbf{e}_{sij}\}$ into account, the new GNN method updates both node and edge embeddings simultaneously:

$$\left\{\mathbf{h}_{si}^{(k)}\right\} = f\left(\left\{\mathbf{h}_{si}^{(k-1)}\right\}, \left\{\mathbf{e}_{sij}^{(k-1)}\right\}, \mathbf{A}\right), \quad \left\{\mathbf{e}_{sij}^{(k)}\right\} = g\left(\left\{\mathbf{h}_{si}^{(k-1)}\right\}, \left\{\mathbf{e}_{sij}^{(k-1)}\right\}, \mathbf{A}\right)$$

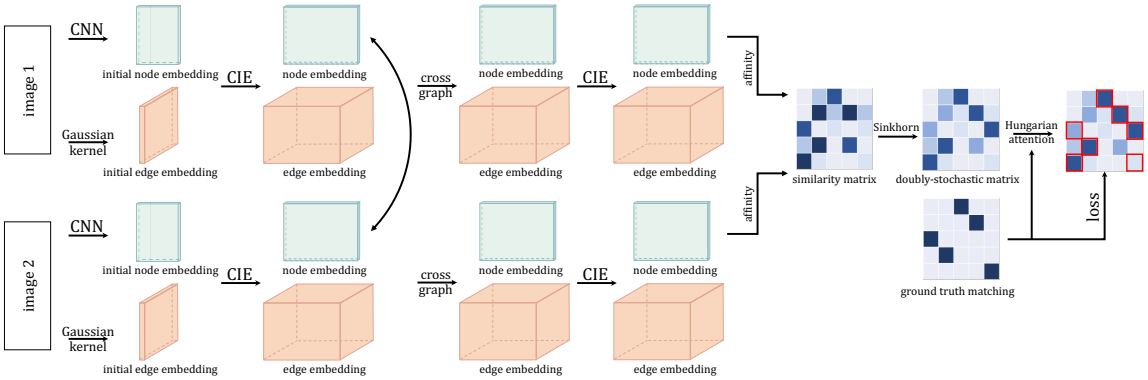



Figure 2: Overall architecture proposed in [14], consisting of channel-independent embedding and Hungarian attention. Picture adopted from [14].

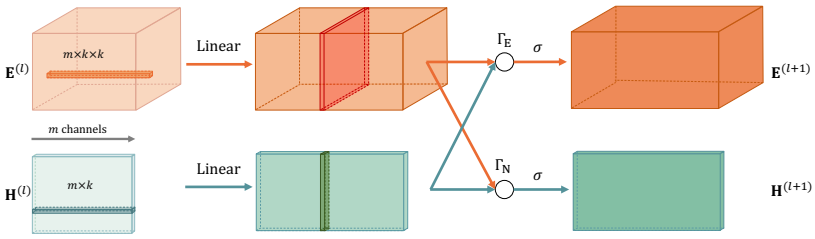



Figure 3: Channel-independent embedding. Picture adopted from [14].

where $\mathbf{A}$ is the adjacency matrix.

**Channel-Independent Embedding.** As is stated in [14], the representation ability of staightforward edge-wise merging is limited. Furthermore, fully connected merging is costy and causes instability for back-propagation. **C**hannel-**I**ndependent **E**mbedding (CIE) is proposed to merge embeddings in a channel-wise fashion

$$\mathbf{h}_{si}^{(k)} = \sigma\left(\sum_{j:(i,j)\in\mathcal{E}_s} \Gamma_N\left(f_{\text{msg-edge}}\left(\mathbf{e}_{sij}^{(k-1)}\right) \circ f_{\text{msg-node}}\left(\mathbf{h}_{sj}^{(k-1)}\right)\right) + \sigma\left(f_{\text{node}}\left(\mathbf{h}_{si}^{(k-1)}\right)\right)\right) \tag{10}$$

$$\mathbf{e}_{sij}^{(k)} = \sigma\left(\Gamma_E\left(f_{\text{msg-edge}}\left(\mathbf{e}_{sij}^{(k-1)}\right) \circ f_{\text{msg-end}}\left(\mathbf{h}_{si}^{(k-1)}, \mathbf{h}_{sj}^{(k-1)}\right)\right) + \sigma\left(f_{\text{msg-edge}}\left(\mathbf{e}_{sij}^{(k-1)}\right)\right)\right) \tag{11}$$

where $\Gamma_N(\cdot \circ \cdot)$, $\Gamma_E(\cdot \circ \cdot)$ are channel-wise operators/functions, and $f_{\text{msg-end}}(\cdot,\cdot)$ is commutative (for undirected graphs). Note that the cross-graph convolution step is still limited in node embeddings. It is also possible to extend the idea for edge embeddings, which require "affinity" values for every edge pair. What keeping us off is the significant computation demand. Here we again stress the "compressibility" of node embeddings for graphs.

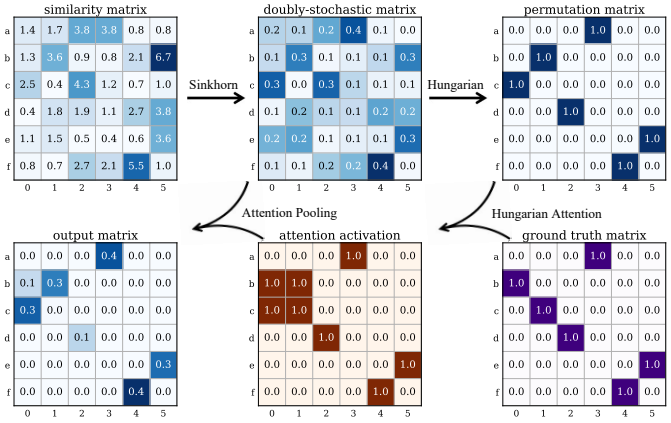



Figure 4: Hungarian attention. Picture adopted from [14].

**Hungarian attention.** Cross-entropy loss prefers samples that are easy to learn in the early training stage and tends to perserve almost binary predictions ( [14]). [14] proposed an improved attention mechanism to better back-propagation and have more unified training and testing interests (we used to train on continuous while test on discrete). They introduce a sparse attention mask as

$$\mathbf{Z} = \text{Atten}\left(\mathbf{X}, \mathbf{X}^{\text{gt}}\right)$$

which is implemented as the union of output binary mask $\mathbf{X}$ and the ground-truth binary mask $\mathbf{X}^{\text{gt}}$. The mask covers all points except for true negatives. They are trivial for optimization especially when we use Sinkhorn (or in other tasks, Softmax) to normalize the outputs. With Hungarian attention, the model is supposed to focus on the performance at ground truth matching and the digits that hinder the matching most. The Hungarian attention loss is defined as

$$\mathcal{H}_{\text{CE}} = -\sum_{i\in\mathcal{V}_1, j\in\mathcal{V}_2} \mathbf{Z}_{ij}\left(\mathbf{X}^{\text{gt}}_{ij}\log\mathbf{S}_{ij} + \left(1-\mathbf{X}^{\text{gt}}_{ij}\right)\log\left(1-\mathbf{S}_{ij}\right)\right)$$

**Soft hungarian attention.** From our observation, the Hungarian attention loss does not improve performance every time. We suspect that the Hungarian attention loss is too strong to be used in the early training stage. The hard attention mechanism makes predictions noisy with submaximums. We thus propose a soft version of Hungarian attention loss, which depress loss from the wide true negatives up to a weight $\lambda_{\text{H}}$. It is defined as

$$\mathcal{H}^{\text{S}}_{\text{CE}} = -\sum_{i\in\mathcal{V}_1, j\in\mathcal{V}_2} \left(\mathbf{Z}_{ij} + \lambda_{\text{H}} \sim \mathbf{Z}_{ij}\right)\left(\mathbf{X}^{\text{gt}}_{ij}\log\mathbf{S}_{ij} + \left(1-\mathbf{X}^{\text{gt}}_{ij}\right)\log\left(1-\mathbf{S}_{ij}\right)\right)$$

where $\sim$ is the bitwise NOT operator. Though formulated from totally different aspect, the soft Hungarian attention loss is computationally equivalent to a weighted sum of the original cross-entropy loss and Hungarian attention loss.

# 3 Experiments

We have done various experiments to compare the mentioned methods, to verify components, and to search for good hyperparameters.

## 3.1 Dataset

We follow tutorials from Pygmtools [11] to take WillowObject [4] as our main dataset. WillowObject is a dataset of images annotated with keypoints. There are altogether 5 classes of images, winebottle, car, motorbike, duck, and face. As is shown in Table 1, face is the easiest class while duck is especially challenging. For fear of overfitting, the train-test set split is conventional so that each class has a balanced number of instances in the train set, resulting in a test set much larger than the train one. And to make up the inbalance of the test set, the overall mean accuracy is calcualted at the level of classes rather than instances.

## 3.2 Methods Comparison

We conduct experiments on PCA-GM, IPCA-GM, CIE, and their combination with Hungarian attention loss (i.e., CIE-H). In Table 1, we report their performance on WillowObject. Among all methods, CIE has the best performance. Despite more refinement steps, IPCA-GM does not beat PCA-GM, which may be caused by longer back-propagation paths.

| Method | Winebottle | Car | Motorbike | Duck | Face | Mean |
|---|---|---|---|---|---|---|
| PCA-GM-w/oFT | 89.32% | 86.47% | 91.26% | 86.32% | 99.92% | 90.66% |
| PCA-GM | 93.90% | 89.89% | 94.47% | 91.68% | **100.00**% | 93.99% |
| PCA-GM-H | 92.62% | 89.42% | 90.74% | 91.68% | **100.00**% | 92.89% |
| IPCA-GM | **95.35**% | 90.32% | 91.05% | 87.03% | 99.97% | 92.74% |
| IPCA-GM-H | 93.19% | 88.00% | 90.79% | 85.24% | 99.91% | 91.43% |
| CIE | 93.65% | **95.58**% | **95.53**% | **92.80**% | 99.17% | **95.34**% |
| CIE-H | 90.39% | 91.05% | 94.95% | 88.21% | 99.93% | 92.90% |

Table 1: Test accuracies of different methods on WillowObject dataset. -w/oFT means the method freezes the pretrained backbone without finetuning, -H means Hungarian attention.

### 3.3 Ablation Study

**Backbone finetuning.** We use pretrained 2D convolutional backbone (VGG16 [9]) for feature extraction. Then it is a natural question whether to further finetune the backbone while training the LAPGM model. In Table 1 we see an overall performance gain with finetuing. So, LAPGM methods would benefit from finetuning learnable feature extractors in an end-to-end style.

**Edge embeddings.** We also see PCA-GM as a variant of CIE with no edge embeddings. As is shown in Table 1, the edge embeddings significantly boost the performance of our LAPGM model, especially for objects with more rigid and templated structures, i.e., car and motorbike.

**Hungarian attention.** Though is described to improve performance of all methods in [14], Hungarian does not necessarily lead to better results in our experiments (see methods with and without -H in Table 1). We also dig deeper into soft Hungarian attention based on CIE. The results are reported in Table 2. Suprisingly, we find that the model learn easier classes (face) better with Hungarian attention, trading off performance on harder classes. And it remains to be seen whether this phenomenon is restricted to WillowObject.

| $\lambda_{\mathrm{H}}$ | Winebottle | Car | Motorbike | Duck | Face | Mean |
|---|---|---|---|---|---|---|
| 0 ($\mathcal{L}_{\mathrm{CE}}$) | 93.65% | **95.58**% | 95.53% | **92.80**% | 99.17% | **95.34**% |
| 0.1 | 93.42% | 89.53% | 96.79% | 89.72% | 99.73% | 93.84% |
| 0.3 | 93.00% | 91.47% | 95.47% | 90.69% | 99.77% | 94.08% |
| 0.5 | 92.32% | 91.84% | 93.89% | 90.53% | 99.85% | 93.69% |
| 0.7 | 90.82% | 88.73% | **96.05**% | 90.76% | 99.68% | 93.21% |
| 0.9 | **94.56**% | 90.47% | 92.21% | 89.08% | **99.97**% | 93.26% |
| 1 ($\mathcal{H}_{\mathrm{CE}}$) | 90.39% | 91.05% | 94.95% | 88.21% | 99.93% | 92.90% |

Table 2: Test accuracies of different level of soft Hungarian attention based on CIE.

### 3.4 Hyperparameter Search

**Weight decay.** Since the test accuracies are obviously worse than train accuracies, there should be some overfitting issues. To prevent overfitting, we choose to impose some regulations on the parameters so as to make the model simple and robust, which is called weight decay. From results shown in Table 3, we conclude that with proper weight decay, the robustness grows while the model capacity does not fall much.

**Optimizer.** We have experimented with both AdamW and SGD optimizers. As the results shown in Table 4, AdamW stably optimize the model to a better minima than SGD.

| Weight Decay | Winebottle | Car | Motorbike | Duck | Face | Mean |
|---|---|---|---|---|---|---|
| 0 | **94.74%** | **93.58%** | 95.26% | 90.09% | **100.00%** | **94.74%** |
| $1e-3$ | 91.21% | 88.58% | **98.21%** | 90.30% | 99.99% | 93.66% |
| $5e-3$ | 92.19% | 92.11% | 93.47% | **91.84%** | **100.00%** | 93.92% |
| $1e-2$ | 93.90% | 89.89% | 94.47% | 91.68% | **100.00%** | 93.99% |
| $5e-2$ | 91.17% | 90.05% | 91.74% | 86.57% | 99.99% | 91.74% |
| $1e-1$ | 92.78% | 91.37% | 94.37% | 89.82% | 99.97% | 93.66% |

Table 3: Test accuracies of PCA-GM model with weight decay.

| Optimizer | Winebottle | Car | Motorbike | Duck | Face | Mean |
|---|---|---|---|---|---|---|
| AdamW | **93.90%** | 89.89% | **94.47%** | **91.68%** | **100.00%** | **93.99%** |
| SGD | 91.17% | **90.05%** | 91.74% | 86.57% | 99.99% | 91.74% |

Table 4: Test accuracies of PCA-GM model with different optimizers.

**Learning rate scheduler.** Several learning rate schedulers are tried. From Table 5, we find that different learning rate schedulers have similar performances in our task.

| LR Scheduler | Winebottle | Car | Motorbike | Duck | Face | Mean |
|---|---|---|---|---|---|---|
| None | **89.52%** | 86.05% | **92.58%** | 86.76% | 99.93% | **90.97%** |
| StepLR | 88.85% | 86.32% | 91.79% | 85.15% | **99.95%** | 90.41% |
| CosineAnnealing | 89.16% | **87.68%** | 89.68% | **87.40%** | 99.93% | 90.77% |

Table 5: Test accuracies of PCA-GM model with different learning rate schedulers.

**More details.** For learning rate, several candidate values are tried at an early stage of the project. Since the performance is rather high and AdamW is not so sensitive to learning rate, a fixed value of $1e-4$ is finally picked for all experiments with no further tuning.

# 4 Implementation and Feedbacks

## 4.1 Trainer

We aim to create a trainer which is modular, robust, and visible. It shall work fine with LAPGM models from Pygmtools [11]. Even more, users can migrate it to any other deep learning task (better with Jittor [7]) painlessly.

**Modularity.** For better reusability and customizability, we build the `Trainer` class in a rather modular way. We make every module quite replacable, including `model`, `optimizer`, `lr_scheduler`, data loaders, benchmarks, and even per-batch task, per-epoch task, evaluation, etc.. For tidiness, we do not implement some functions as callable arguments, but they are also easy to substitute.

**Robustness.** Our trainer is robust and automated. It would log in file the training and testing process and save checkpoints automatically. Besides the main model, all states including training progress can be fully recorded (up to the config) and automatically resumed. Users need not to worry about accidents during training.

**Visibility.** The training process and all informations are printed in a human-friendly way. You can supervise the progress with ease. By the way, all training statics like loss and metrics are connected to TensorBoard [1], ready for investigating and further processing.

### 4.2 Dataset

The dataset is implemented based on `benchmark` class from Pygmtools [11] and `Dataset` class from Jittor [7], with which data loading and preprocessing are easy. For better compatibility, we have implemented Delaunay triangulation with both SciPy [12] and OpenCV [3]. Furthermore, since `benchmark` shuffles the order of keypoints each time loading data, we adviced that the cached ground truth should be refreshed at each shuffle for coherence, which has been adopted in newer versions of Pygmtools. As an abstract, the dataset is iterable. At each iteration, a batch (`dict`) of data is provided.

**WillowObject.** This dataset is quite standard. We can directly use the universal implementation to get it smoothly involved in training.

**PascalVOC.** Numbers of keypoints varies from instance to instance in PascalVOC [2, 6]. So we specially provide batch support based on padding and support for cases in which keypoints are less than 3. For batch support, we rewrite `collate_batch` to make all shapes correct. To fit the interface of `Benchmark.eval`, we perform a matrix-to-list pipeline while testing. We have intended to perform some quantitative experiments on PascalVOC, but a CUDA error from Jittor [7] was reported for a long period and the dataset itself is too big to do full experiments. Consequently, only a compatibility test is done, which shows that our pipeline works with the dataset. (P.S., as is mentioned in the presentation, it would be more user-friendly if the `voc2011_pairs.npz` file is downloaded automatically together with the main data.)

### 4.3 Model

We have integrated multiple mentioned methods in one universal `LAPGM` class. It has two main modules, a 2D backbone module called `extractor` and a group matching module consisting of `gm_fn` and `gm_model` based on interfaces provided by Pygmtools [11]. The model first extract feature from images and construct a graph from keypoints and the corresponding features. Then it passes the data to the graph matching method to get predictions. Finally, it calculates the loss if target is provided.

**Efficiency.** During data processing, we sucessfully replace all possible Python loop with efficient batched operations provided by Jittor [7], so as to shorten inference time.

**Hungarian attention.** Although it might be straightforward to implement Hungarian attention by reimplementing the permutation loss, we choose to use gradient hook instead. On one hand, gradient hook is orthognal to the loss on which Hungarian attention is applied. We can clearly see the weight applied to each element and replace the loss function without reimplementation. On the other hand, we are able to make more meaningful comparisons on loss if needed since they are calculated with the same equation.

## 5 Acknowledgements

Thanks to Dr. Junchi Yan for lectures and advice. Thanks to TAs (especially Runzhong Wang) for advice, instructions, and updates on Pygmtools [11].

# References


[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] L. Bourdev and J. Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *International Conference on Computer Vision*, pages 1365–1372. IEEE, 2009.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Minsu Cho, Karteek Alahari, and Jean Ponce. Learning graphs to match. In *International Conference on Computer Vision*, pages 25–32, 2013.

[5] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[6] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2010.

[7] Shi-Min Hu, Dun Liang, Guo-Ye Yang, Guo-Wei Yang, and Wen-Yang Zhou. Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Science China Information Sciences*, 63(222103):1–21, 2020.

[8] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[10] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

[11] ThinkLab. Pygmtools. https://github.com/Thinklab-SJTU/pygmtools, 2022.

[12] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[13] Runzhong Wang, Junchi Yan, and Xiaokang Yang. Combinatorial learning of robust deep graph matching: an embedding based approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[14] Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Learning deep graph matching with channel-independent embedding and hungarian attention. In *International Conference on Learning Representations*, 2020.