学习会

START

赤潮预测项目

模型参数

	原模型		当前模型
隐层单元数	5	→	32
优化器	Nadam	→	Adam
BatchSize	100	→	128
Stateful	True	→	False

01 模型编写

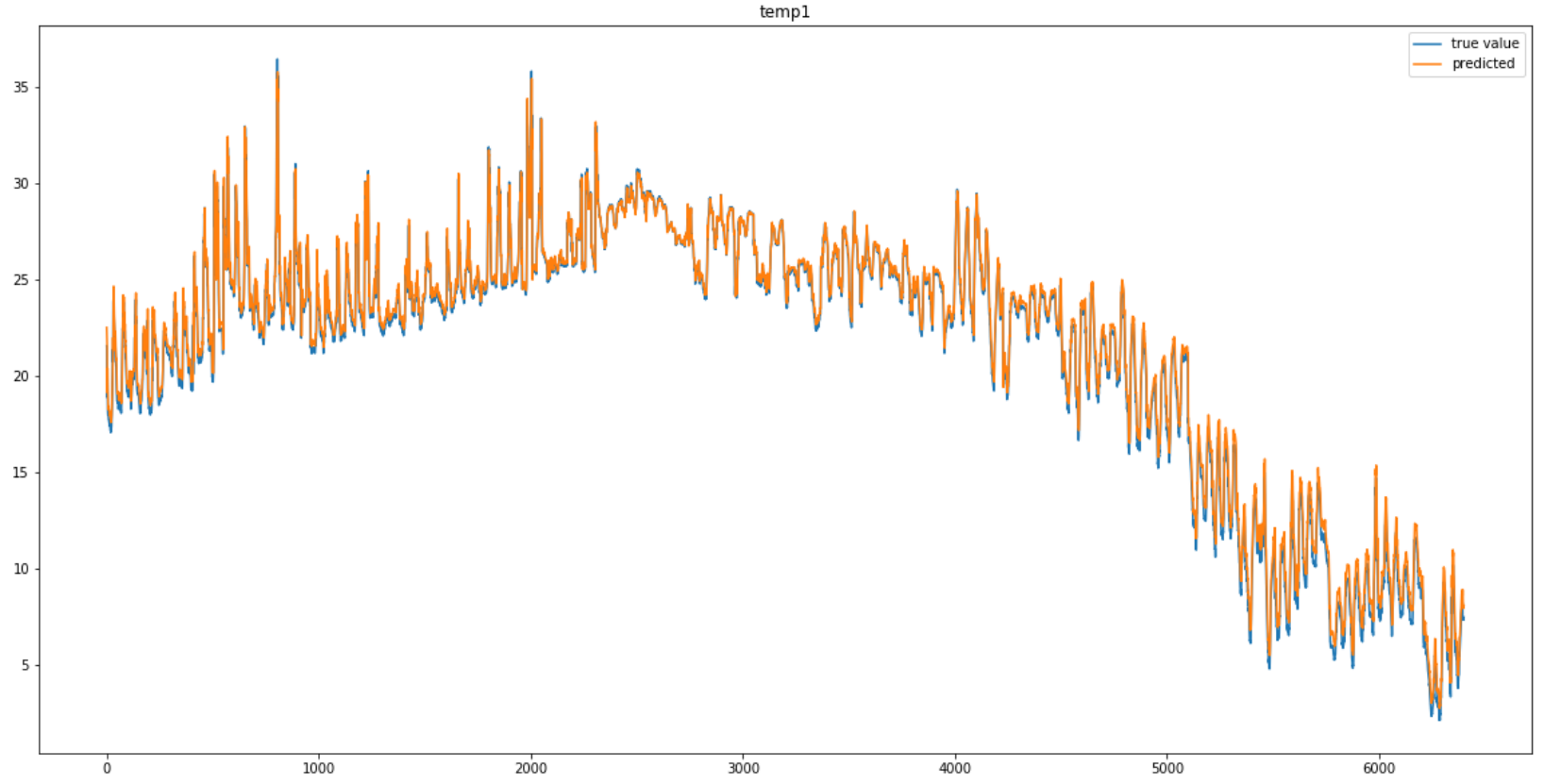
In [176]:
```
class Model(nn.Module):
    def __init__(self, input_size, hidden_size, output_dim):
        super(Model, self).__init__()
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size, batch_first=True)
#         self.hidden = (torch.zeros(1, batch_size, hidden_size), torch.zeros(1, batch_size, h
        self.linear = nn.Linear(hidden_size, output_dim)
    def forward(self, x):
        x, _ = self.lstm(x, None)
        last_output = x[:, -1, :]
        x = self.linear(last_output)
        out = F.relu(x, inplace=True)
        return out
```

In [154]:
```
model = Model(1, 32, 3).to(device)
```

In [183]:
```
criterion = nn.MSELoss().to(device)
ops = optim.Adam(model.parameters(), lr=1e-5)
```

模型编写

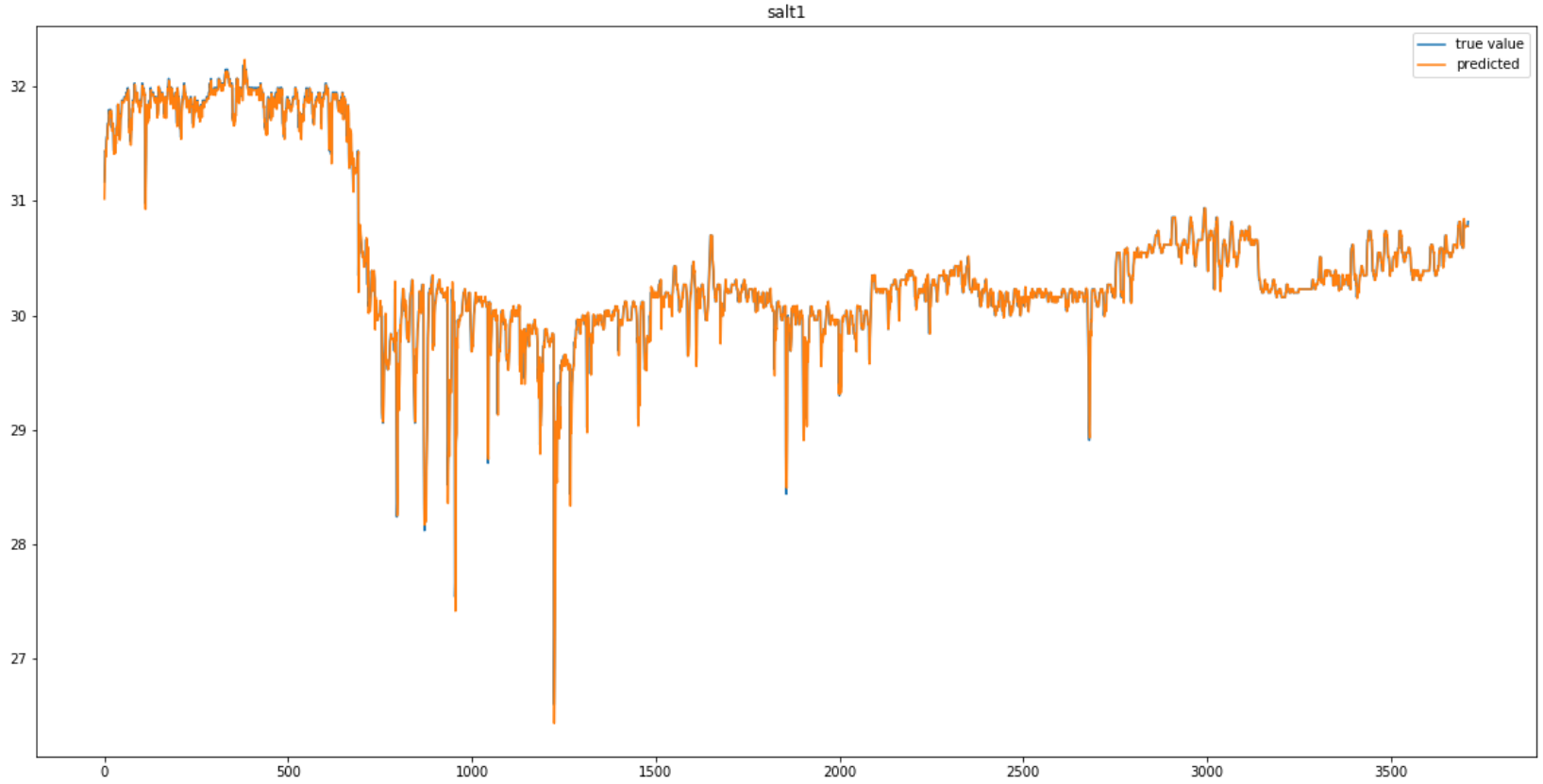
```
for epoch in range(300):
    total_loss = 0.0
    for batch in range(num_batch):
        model.train()
        x = x_train[batch*batch_size:(batch+1)*batch_size]
        y = y_train[batch*batch_size:(batch+1)*batch_size]
        logits = model(x)
        loss = criterion(logits, y)
        ops.zero_grad()
        loss.backward()
        ops.step()
        total_loss += loss
    model.eval()
    y_pre = model(x_test)
    test_loss = criterion(y_pre, y_test)
    print('epoch:%d train_loss:%e, val_loss:%e' %(epoch, total_loss.item()/(len(x_train)/batch_size), test_loss.
```



气温

1号13-17年训练18年预测

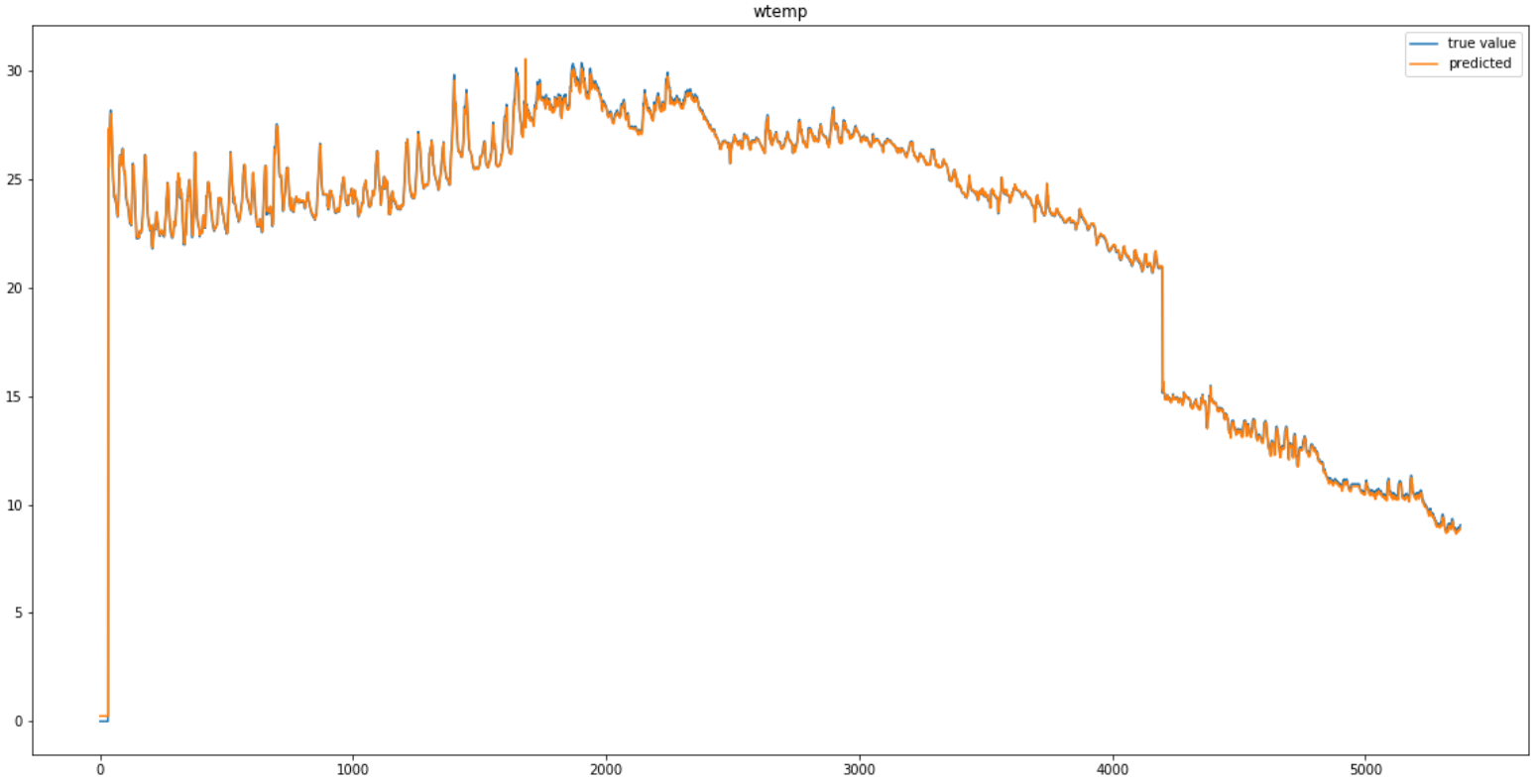
蓝色为真实值，橙色为预测值



盐度

1号13-17训练18预测

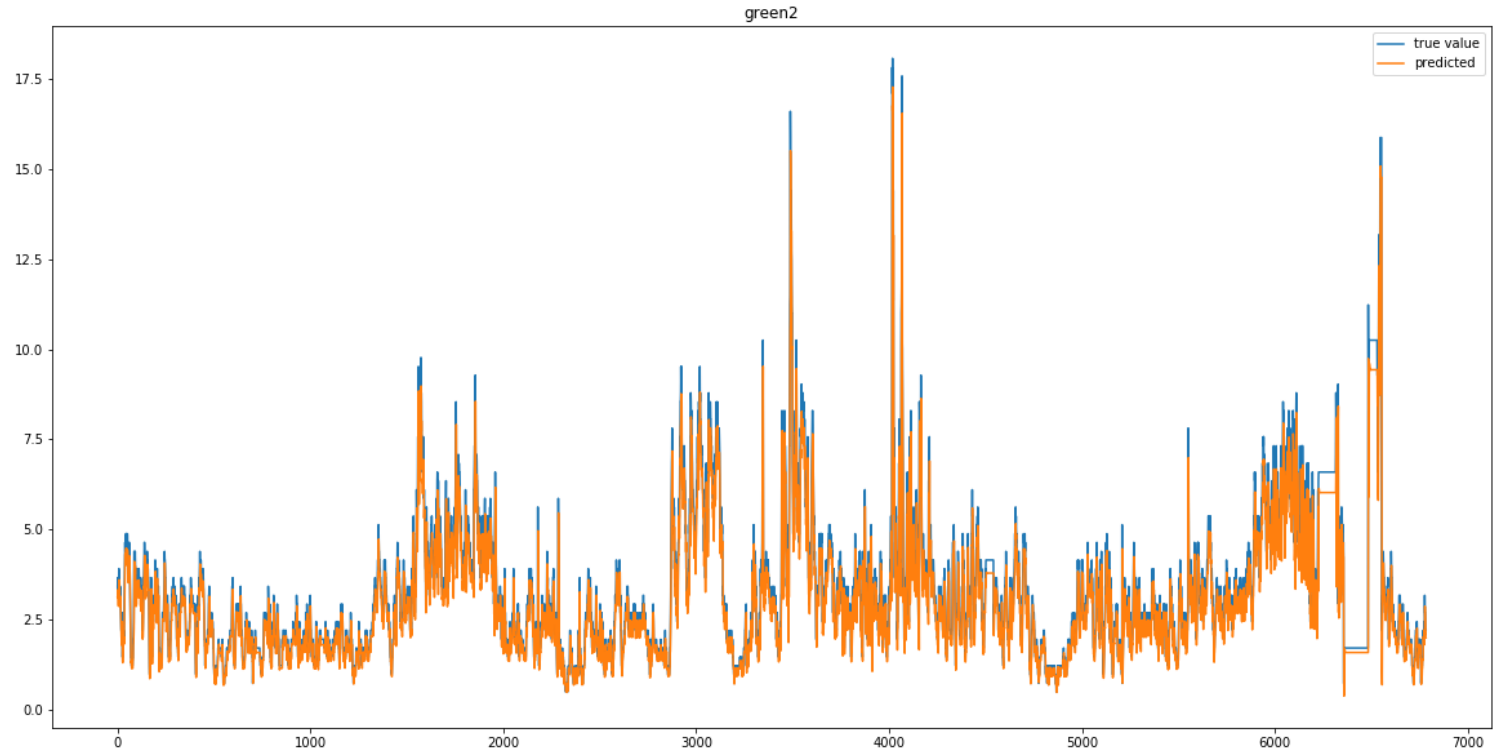
蓝色为真实值，橙色为预测值



水温

1号13-17年训练18年预测

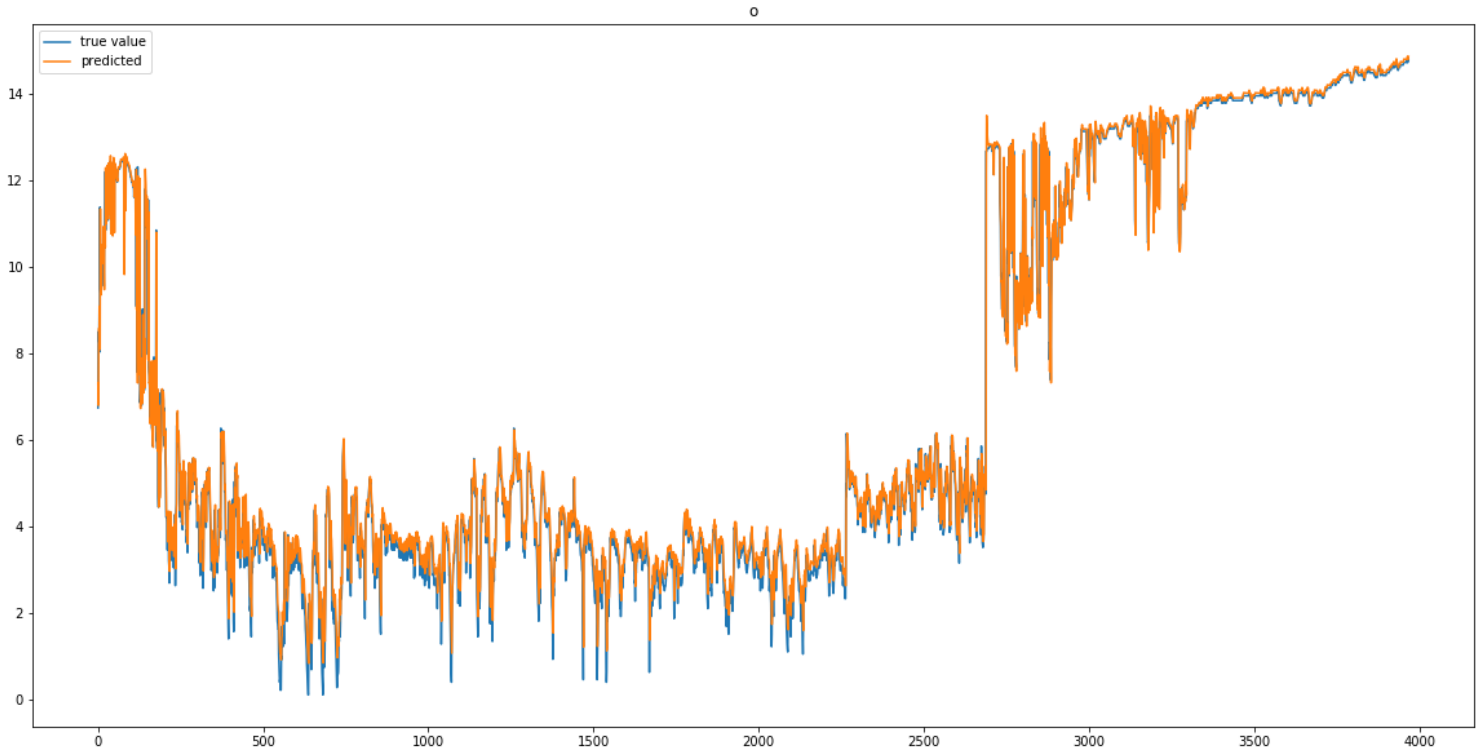
蓝色为真实值，橙色为预测值



叶绿素

1号13-17训练18预测

蓝色为真实值，橙色为预测值



溶解氧

1号13-17训练18预测

蓝色为真实值，橙色为预测值

学习与分享

02

论文

1.The Graph Neural Network Model

2. Convolutional LSTM Network

02

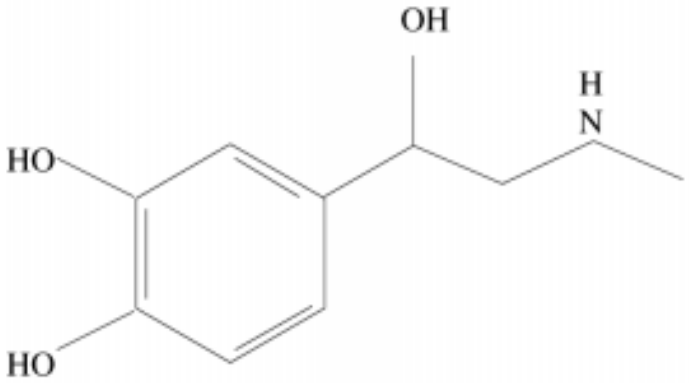
The Graph Neural Network Model

I. Introduction

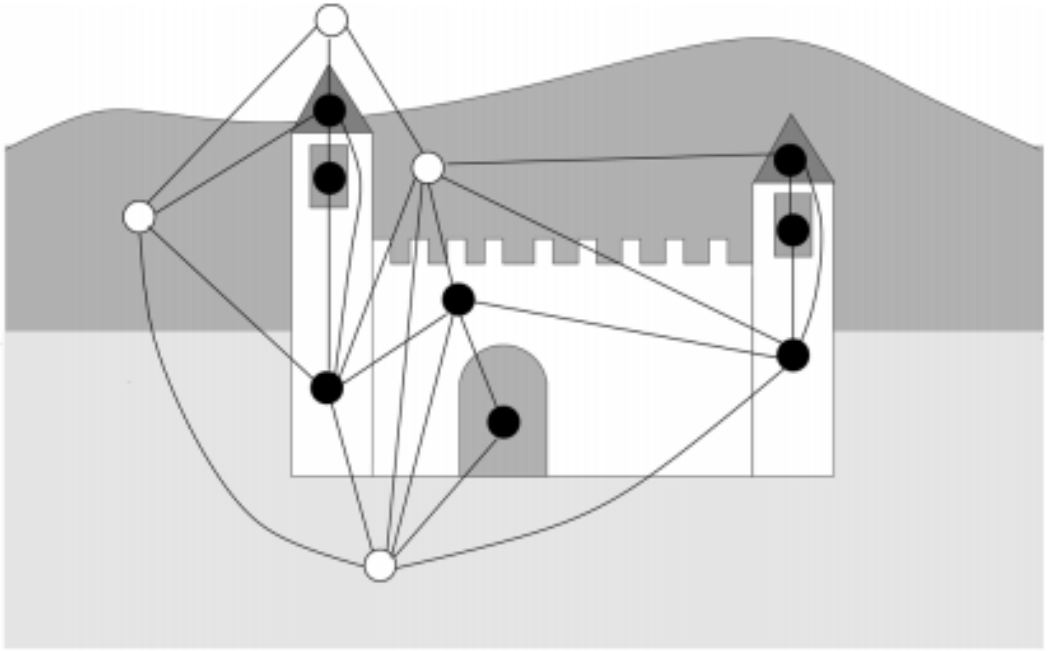
DATA can be naturally represented by graph structures in several application areas, including proteomics [1], image analysis [2], scene description [3], [4], software engineering [5], [6], and natural language processing [7]. The simplest kinds of graph structures include single nodes and sequences. But in several applications, the information is organized in more complex graph structures such as trees, acyclic graphs, or cyclic graphs. Traditionally, data relationships exploitation has been the subject of many studies in the community of inductive logic programming and, recently, this research theme has been evolving in different directions [8], also because of the applications of relevant concepts in statistics and neural networks to such areas (see, for example, the recent workshops [9]–[12]).

02

The Graph Neural Network Model



(a)



(b)

02

The Graph Neural Network Model

$$\begin{aligned} \boldsymbol{x}_n &= f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]}) \\ \boldsymbol{o}_n &= g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n) \end{aligned} \tag{1}$$

where $\boldsymbol{l}_n$, $\boldsymbol{l}_{\mathrm{co}[n]}$, $\boldsymbol{x}_{\mathrm{ne}[n]}$, and $\boldsymbol{l}_{\mathrm{ne}[n]}$ are the label of n, the labels of its edges, the states, and the labels of the nodes in the neighborhood of n, respectively.

Let $\boldsymbol{x}$, $\boldsymbol{o}$, $\boldsymbol{l}$, and $\boldsymbol{l_N}$ be the vectors constructed by stacking all the states, all the outputs, all the labels, and all the node labels, respectively. Then, (1) can be rewritten in a compact form as

$$\begin{aligned} \boldsymbol{x} &= F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}) \\ \boldsymbol{o} &= G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l_N}) \end{aligned} \tag{2}$$

02

The Graph Neural Network Model

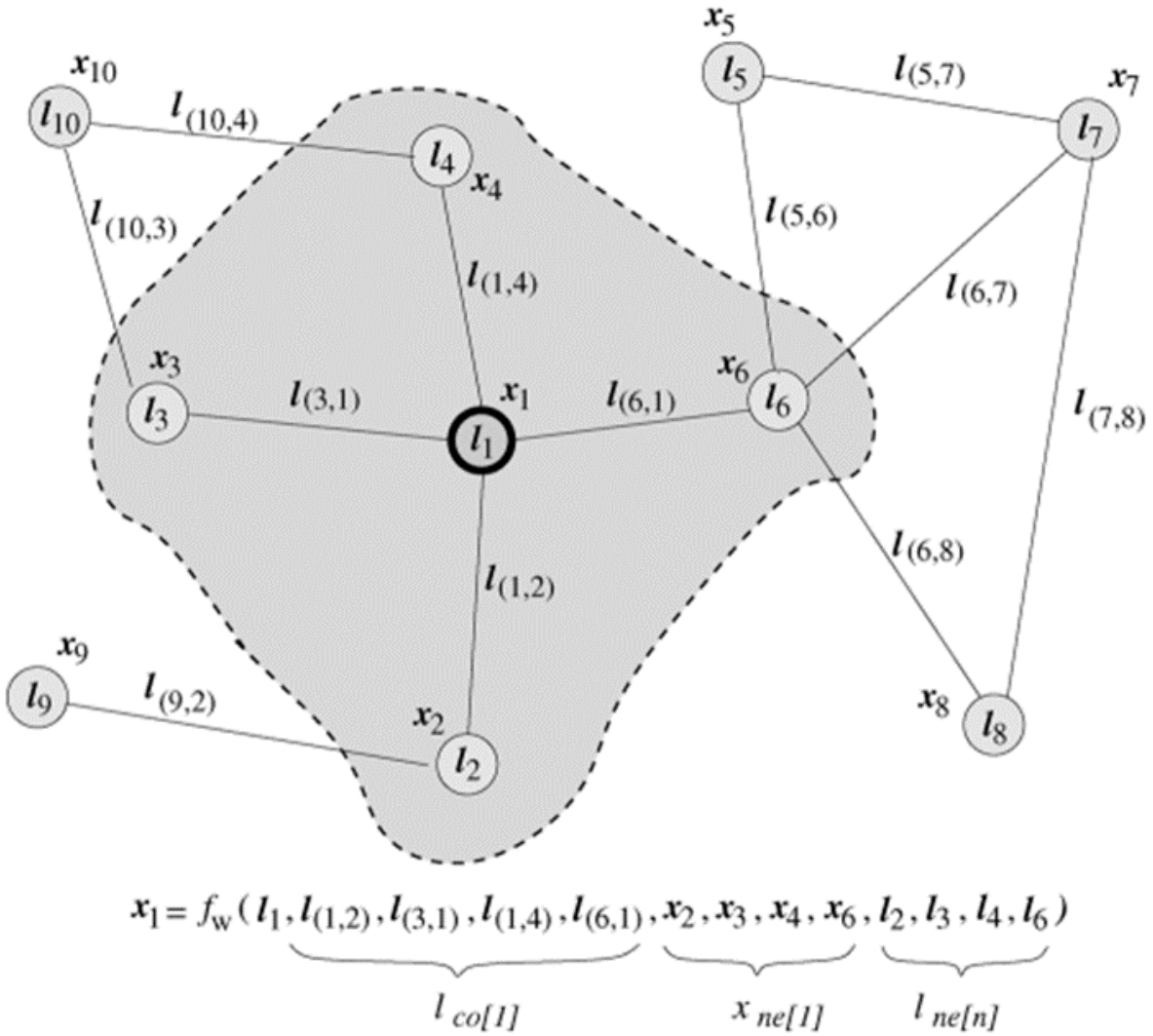
Note that (1) makes it possible to process both positional and nonpositional graphs. For positional graphs, $f_{\boldsymbol{w}}$ must receive the positions of the neighbors as additional inputs. In practice, this can be easily achieved provided that information contained in $\boldsymbol{x}_{\mathrm{ne}[n]}$, $\boldsymbol{l}_{\mathrm{co}[n]}$, and $\boldsymbol{l}_{\mathrm{ne}[n]}$ is sorted according to neighbors' positions and is properly padded with special null values in positions corresponding to nonexisting neighbors. For example, $\boldsymbol{x}_{\mathrm{ne}[n]} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M]$, where $M = \max_{n,u} \nu_n(u)$ is the maximal number of neighbors of a node; $\boldsymbol{y}_i = \boldsymbol{x}_u$ holds, if u is the ith neighbor of n ($\nu_n(u) = i$); and $\boldsymbol{y}_i = \boldsymbol{x}_0$, for some predefined null state $\boldsymbol{x}_0$, if there is no ith neighbor.

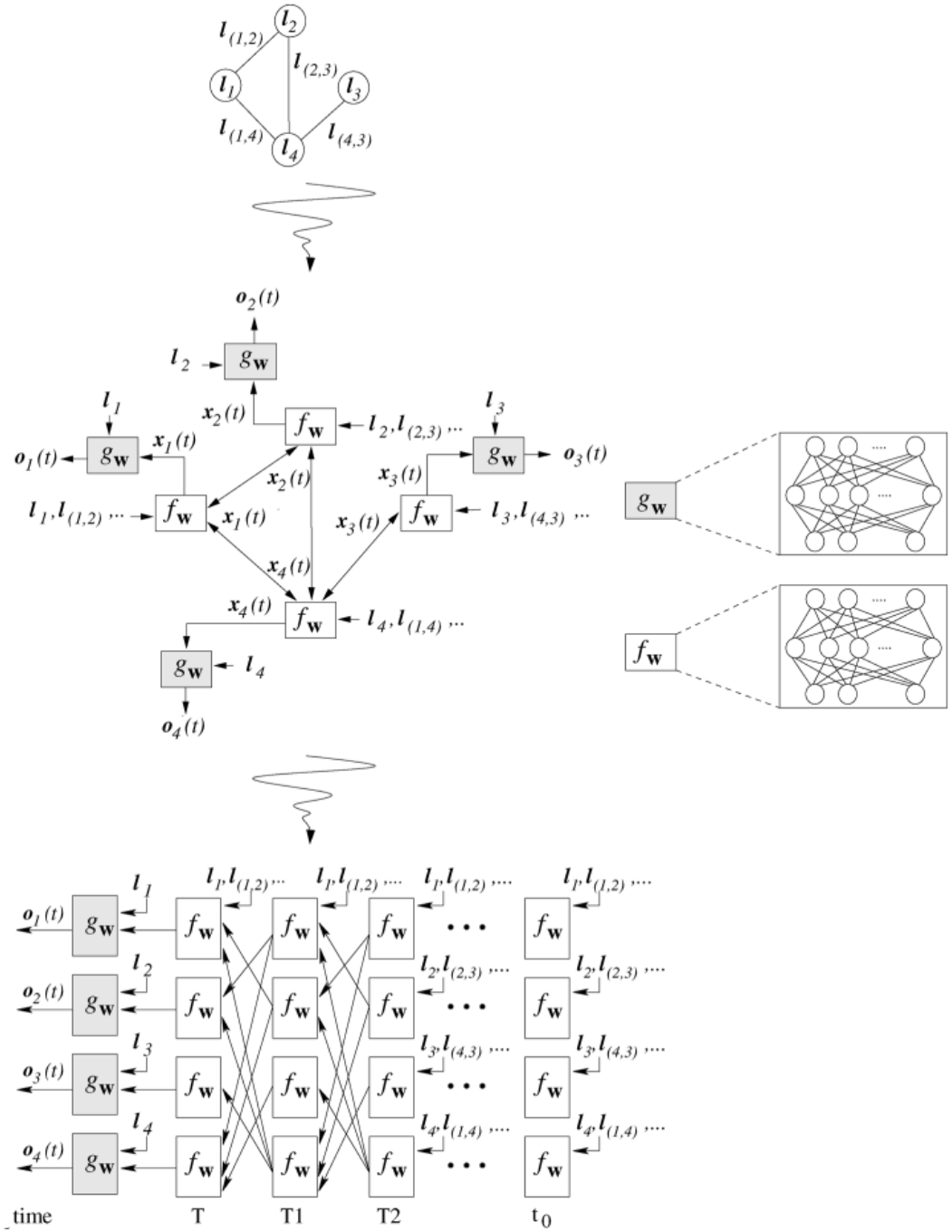
However, for nonpositional graphs, it is useful to replace function $f_{\boldsymbol{w}}$ of (1) with

$$\boldsymbol{x}_n = \sum_{u \in \mathrm{ne}[n]} h_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u), \qquad n \in \boldsymbol{N} \tag{3}$$

02

The Graph Neural Network Model




$l_{(1,2)}$
l_2
$l_{(2,3)}$
l_1
l_3
$l_{(1,4)}$
l_4
$l_{(4,3)}$
$o_2(t)$
g_w
l_2
$x_2(t)$
f_w
$l_2, l_{(2,3)}$ ...
l_3
l_1
$x_1(t)$
$o_1(t)$
g_w
$x_2(t)$
$x_3(t)$
g_w
$o_3(t)$
$l_1, l_{(1,2)}$ ...
f_w
$x_1(t)$
$x_3(t)$
f_w
$l_3, l_{(4,3)}$ ...
$x_4(t)$
$x_4(t)$
f_w
$l_4, l_{(1,4)}$ ...
g_w
l_4
$o_4(t)$
g_w
f_w
time
T
T1
T2
t_0

02

The Graph Neural Network Model

B. Computation of the State

Banach's fixed point theorem [53] does not only ensure the existence and the uniqueness of the solution of (1) but it also suggests the following classic iterative scheme for computing the state:

$$\boldsymbol{x}(t+1) = F_{\boldsymbol{w}}(\boldsymbol{x}(t), \boldsymbol{l}) \tag{4}$$

where $\boldsymbol{x}(t)$ denotes the tth iteration of $\boldsymbol{x}$. The dynamical system (4) converges exponentially fast to the solution of (2) for any initial value $\boldsymbol{x}(0)$. We can, therefore, think of $\boldsymbol{x}(t)$ as the state that is updated by the transition function $F_{\boldsymbol{w}}$. In fact, (4) implements the Jacobi iterative method for solving nonlinear equations [55]. Thus, the outputs and the states can be computed by iterating

$$\begin{aligned} \boldsymbol{x}_n(t+1) &= f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}_{\mathrm{ne}[n]}) \\ \boldsymbol{o}_n(t) &= g_{\boldsymbol{w}}(\boldsymbol{x}_n(t), \boldsymbol{l}_n), \qquad n \in \boldsymbol{N}. \end{aligned} \tag{5}$$

1) *Linear (nonpositional) GNN.* Equation (3) can naturally be implemented by

$$h_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{x}_u, \boldsymbol{l}_u) = \boldsymbol{A}_{n,u}\boldsymbol{x}_u + \boldsymbol{b}_n \tag{12}$$

where the vector $\boldsymbol{b}_n \in \mathbb{R}^s$ and the matrix $\boldsymbol{A}_{n,u} \in \mathbb{R}^{s\times s}$ are defined by the output of two feedforward neural networks (FNNs), whose parameters correspond to the parameters of the GNN. More precisely, let us call *transition network* an FNN that has to generate $\boldsymbol{A}_{n,u}$ and *forcing network* another FNN that has to generate $\boldsymbol{b}_n$. Moreover, let $\phi_{\boldsymbol{w}} : \mathbb{R}^{2l_N+l_E} \to \mathbb{R}^{s^2}$ and $\rho_{\boldsymbol{w}} : \mathbb{R}^{l_N} \to \mathbb{R}^s$ be the functions implemented by the transition and the forcing network, respectively. Then, we define

$$\boldsymbol{A}_{n,u} = \frac{\mu}{s|\mathrm{ne}[u]|} \cdot \boldsymbol{\Xi} \tag{13}$$

$$\boldsymbol{b}_n = \rho_{\boldsymbol{w}}(\boldsymbol{l}_n) \tag{14}$$

where $\mu \in (0, 1)$ and $\boldsymbol{\Xi} = \mathrm{resize}(\phi_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{l}_u))$ hold, and $\mathrm{resize}(\cdot)$ denotes the operator that allocates the elements of a s^2-dimensional vector into as $s \times s$ matrix. Thus, $\boldsymbol{A}_{n,u}$ is obtained by arranging the outputs of the transition network into the square matrix $\boldsymbol{\Xi}$ and by multiplication with the factor $\mu/s|\mathrm{ne}[u]|$. On the other hand, $\boldsymbol{b}_n$ is just a vector that contains the outputs of the forcing network. Here, it is further assumed that $\|\phi_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{(n,u)}, \boldsymbol{l}_u)\|_1 \leq s$ holds[7]; this can be straightforwardly verified if the output neurons of the transition network use an appropriately bounded activation function, e.g., a hyperbolic tangent. Note that in this case $F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$, where $\boldsymbol{b}$ is the vector constructed by stacking all the $\boldsymbol{b}_n$, and $\boldsymbol{A}$ is a block matrix $\{\bar{\boldsymbol{A}}_{n,u}\}$, with $\bar{\boldsymbol{A}}_{n,u} = \boldsymbol{A}_{n,u}$ if u is a neighbor of n and $\bar{\boldsymbol{A}}_{n,u} = \boldsymbol{0}$ otherwise. Moreover, vectors $\boldsymbol{b}_n$ and matrices $\boldsymbol{A}_{n,u}$ do not depend on the state $\boldsymbol{x}$, but only on node and edge labels. Thus, $\partial F_{\boldsymbol{w}}/\partial \boldsymbol{x} = \boldsymbol{A}$, and, by simple algebra

$$\left\|\frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}\right\|_1 = \|\boldsymbol{A}\|_1 \leq \max_{u\in \boldsymbol{N}} \left(\sum_{n\in \mathrm{ne}[u]} \|\boldsymbol{A}_{n,u}\|_1\right)$$

$$\leq \max_{u\in \boldsymbol{N}} \left(\frac{\mu}{s|\mathrm{ne}[u]|} \cdot \sum_{n\in \mathrm{ne}[u]} \|\boldsymbol{\Xi}\|_1\right) \leq \mu$$

which implies that $F_{\boldsymbol{w}}$ is a contraction map (w.r.t. $\|\cdot\|_1$) for any set of parameters $\boldsymbol{w}$.

2) *Nonlinear (nonpositional) GNN.* In this case, $h_{\boldsymbol{w}}$ is realized by a multilayered FNN. Since three-layered neural networks are universal approximators [67], $h_{\boldsymbol{w}}$ can approximate any desired function. However, not all the parameters $\boldsymbol{w}$ can be used, because it must be ensured that the corresponding transition function $F_{\boldsymbol{w}}$ is a contraction map. This can be achieved by adding a penalty term to (6), i.e.,

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p}\sum_{j=1}^{q_i}(\boldsymbol{t}_{i,j} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2 + \beta L\left(\left\|\frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}\right\|\right)$$

where the penalty term $L(y)$ is $(y-\mu)^2$ if $y > \mu$ and 0 otherwise, and the parameter $\mu \in (0, 1)$ defines the desired contraction constant of $F_{\boldsymbol{w}}$. More generally, the penalty term can be any expression, differentiable w.r.t. $\boldsymbol{w}$, that is monotone increasing w.r.t. the norm of the Jacobian. For example, in our experiments, we use the penalty term $p_{\boldsymbol{w}} = \sum_{i=1}^{s} L(\|\boldsymbol{A}^i\|_1)$, where $\boldsymbol{A}^i$ is the ith column of $\partial F_{\boldsymbol{w}}/\partial \boldsymbol{x}$. In fact, such an expression is an approximation of $L(\|\partial F_{\boldsymbol{w}}/\partial \boldsymbol{x}\|_1) = L(\max_i \|\boldsymbol{A}^i\|_1)$.

MAIN
initialize $\boldsymbol{w}$;
$\boldsymbol{x}$=**Forward**($\boldsymbol{w}$);
repeat
$\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$=BACKWARD($\boldsymbol{x}, \boldsymbol{w}$);
$\boldsymbol{w}$=$\boldsymbol{w} - \lambda \cdot \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;
$\boldsymbol{x}$=FORWARD($\boldsymbol{w}$);
until (a stopping criterion);
return $\boldsymbol{w}$;
end

FORWARD($\boldsymbol{w}$)
initialize $\boldsymbol{x}(0)$, $t = 0$;
repeat
$\boldsymbol{x}(t+1) = F_{\boldsymbol{w}}(\boldsymbol{x}(t), \boldsymbol{l})$;
t=$t+1$;
until $\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\| \leq \varepsilon_f$
return $\boldsymbol{x}(t)$;
end

BACKWARD($\boldsymbol{x}$,$\boldsymbol{w}$)
$\boldsymbol{o} = G_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_N)$;
$\boldsymbol{A} = \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l})$;
$\boldsymbol{b} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{l}_N)$;
initialize $\boldsymbol{z}(0)$, t=0;
repeat
$\boldsymbol{z}(t) = \boldsymbol{z}(t+1) \cdot \boldsymbol{A} + \boldsymbol{b}$;
t=$t-1$;
until $\|\boldsymbol{z}(t-1) - \boldsymbol{z}(t)\| \leq \varepsilon_b$;
$\boldsymbol{c} = \frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{o}} \cdot \frac{\partial G_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}_N)$;
$\boldsymbol{d} = \boldsymbol{z}(t) \cdot \frac{\partial F_{\boldsymbol{w}}}{\partial \boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$;
$\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}} = \boldsymbol{c} + \boldsymbol{d}$;
return $\frac{\partial e_{\boldsymbol{w}}}{\partial \boldsymbol{w}}$;
end

Learning in GNNs consists of estimating the parameter $\boldsymbol{w}$ such that $\varphi_{\boldsymbol{w}}$ approximates the data in the learning data set

$$\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j})|, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G}; \\ n_{i,j} \in \boldsymbol{N}_i; \boldsymbol{t}_{i,j} \in \mathbb{R}^m, 1 \leq i \leq p, 1 \leq j \leq q_i\}$$

where q_i is the number of supervised nodes in $\boldsymbol{G}_i$. For graph-focused tasks, one special node is used for the target ($q_i = 1$ holds), whereas for node-focused tasks, in principle, the supervision can be performed on every node. The learning task can be posed as the minimization of a quadratic cost function

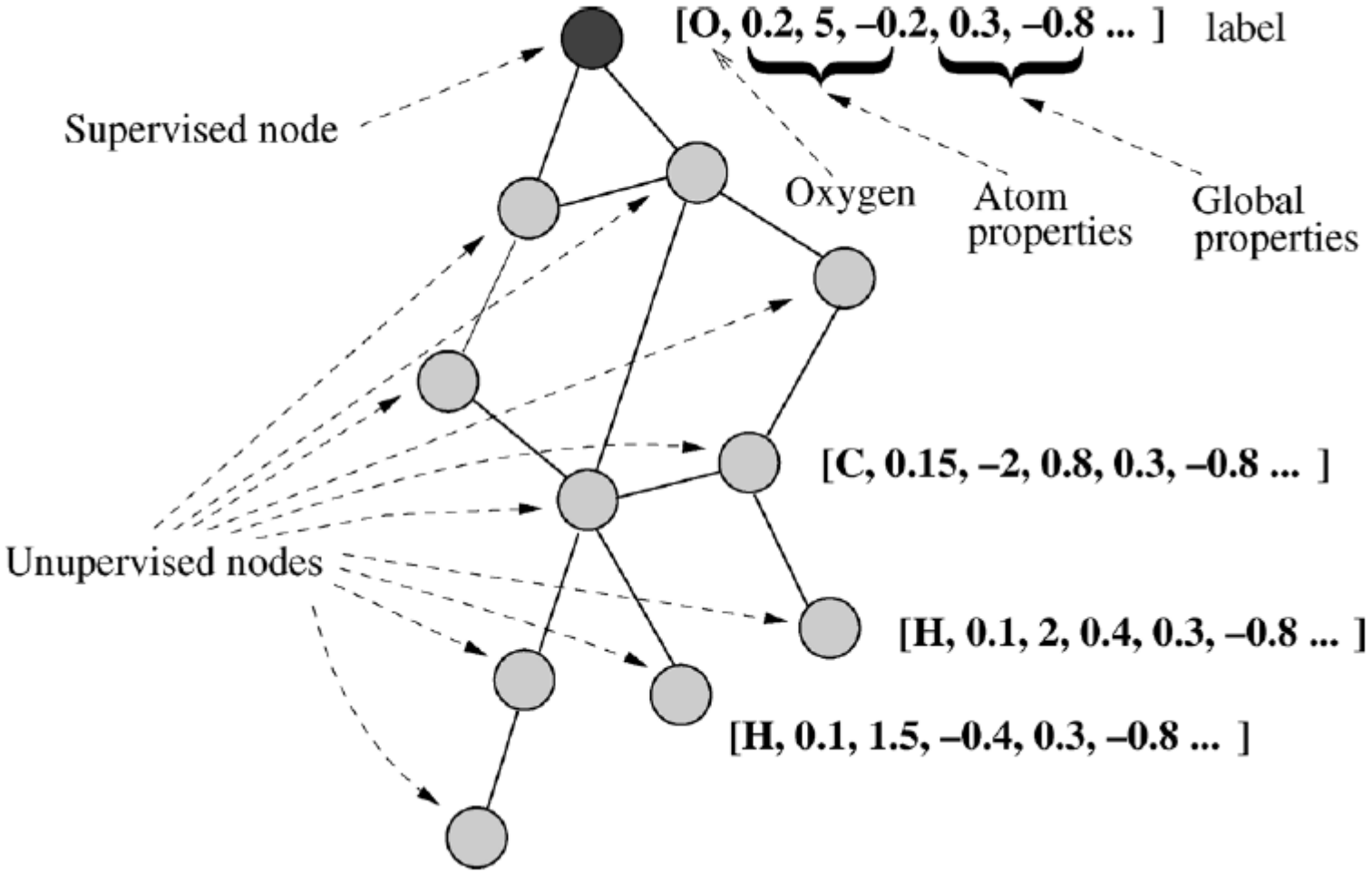
$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (\boldsymbol{t}_{i,j} - \varphi_{\boldsymbol{w}}(\boldsymbol{G}_i, n_{i,j}))^2 . \tag{6}$$

The learning algorithm is based on a gradient-descent strategy and is composed of the following steps.

a) The states $\boldsymbol{x}_n(t)$ are iteratively updated by (5) until at time T they approach the fixed point solution of (2): $\boldsymbol{x}(T) \approx \boldsymbol{x}$.
b) The gradient $\partial e_{\boldsymbol{w}}(T)/\partial \boldsymbol{w}$ is computed.
c) The weights $\boldsymbol{w}$ are updated according to the gradient computed in step b).

02

The Graph Neural Network Model



02

Convolutional LSTM Network

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o) \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
\tag{2}
$$

02

Convolutional LSTM Network

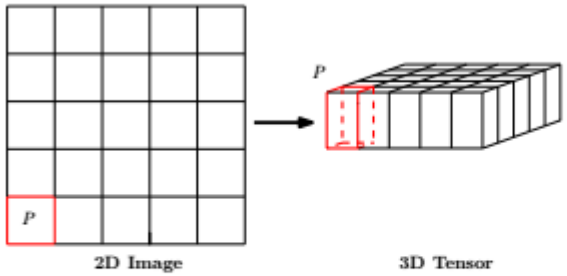


Figure 1: Transforming 2D image into 3D tensor

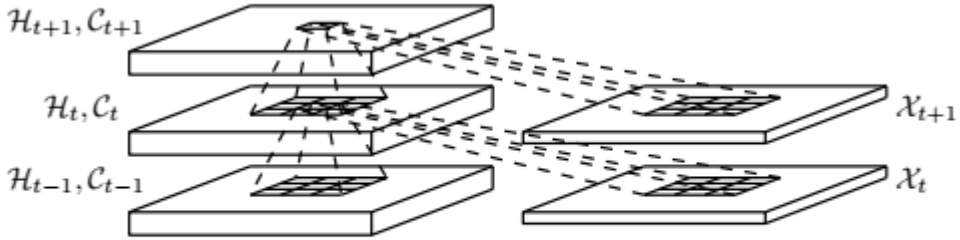


Figure 2: Inner structure of ConvLSTM

$$
\begin{aligned}
i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\
f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\
o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\
\mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t)
\end{aligned}
$$

02

Convolutional LSTM Network

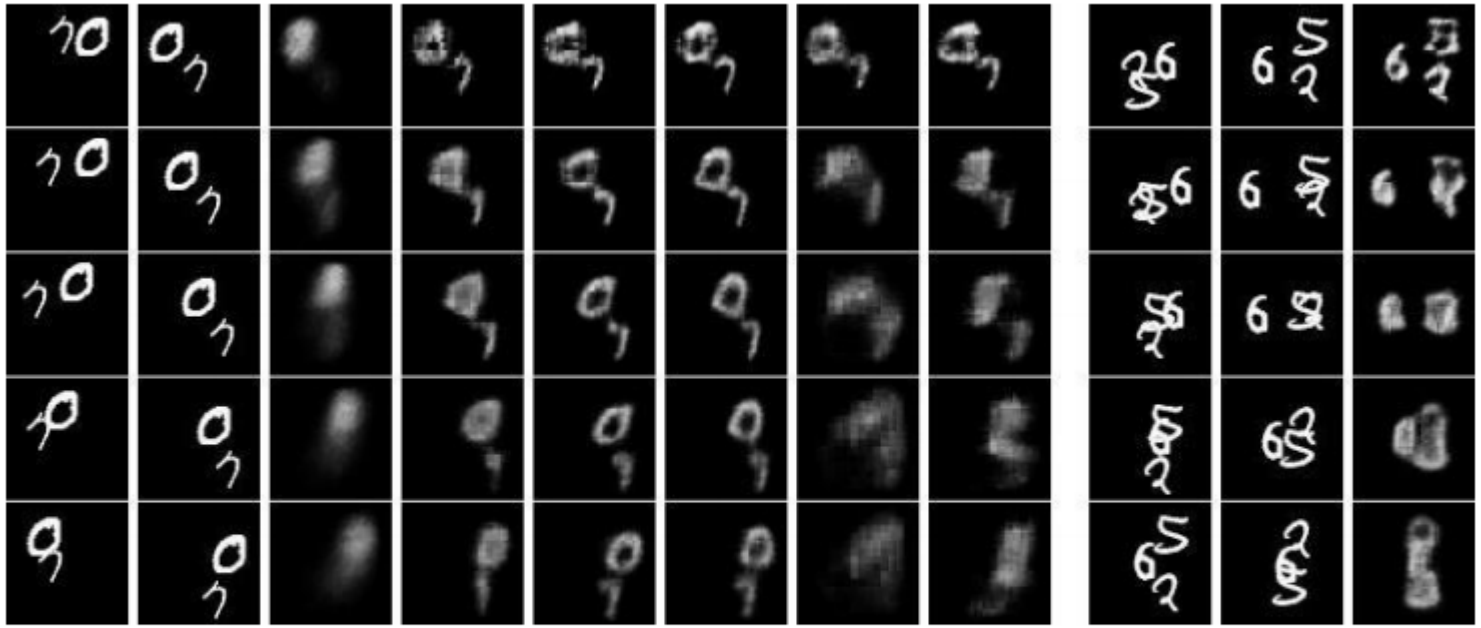


Figure 4: **Left part is an example showing the prediction results of different models.** From left to right: input frames; ground truth; FC-LSTM; 1-layer; 2-layer; 3-layer; 2-layer without convolution; 3-layer without convolution. **Right part is an example showing an "out-of-domain" run.** From left to right: input frames; ground truth; 3-layer. All sequences are sampled with an interval of 2.

02

Convolutional LSTM Network

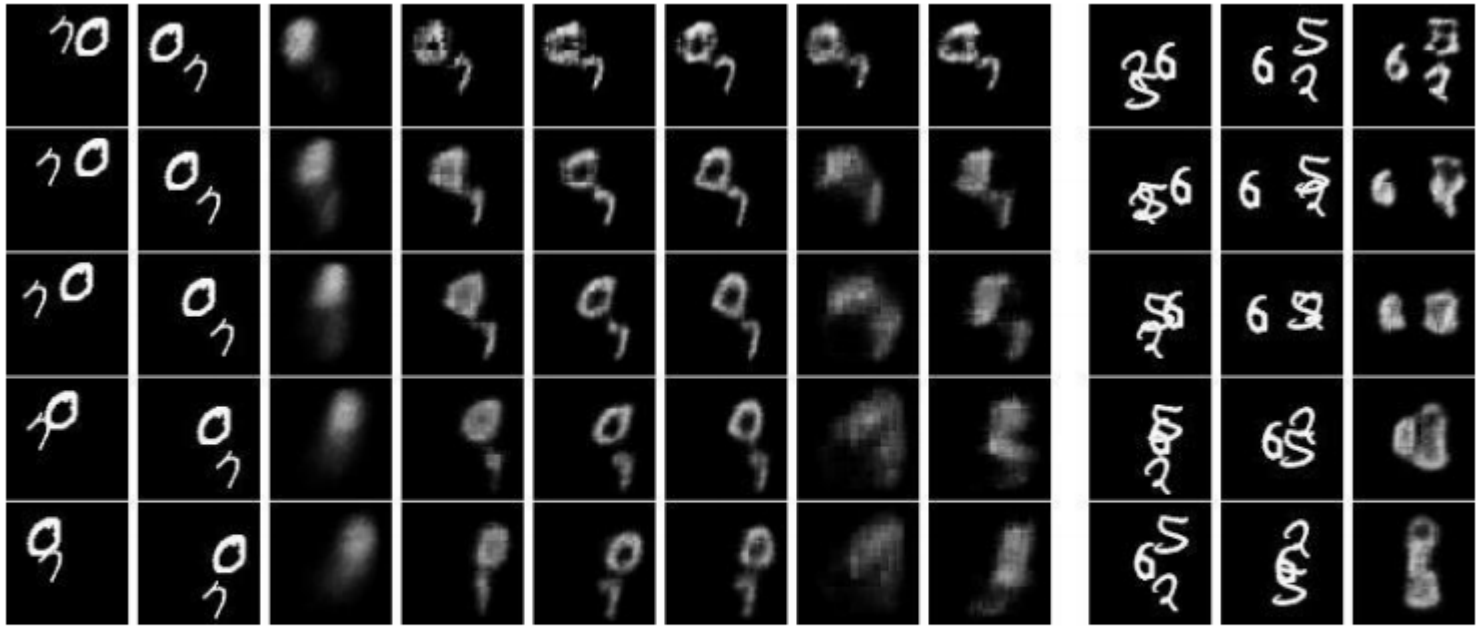
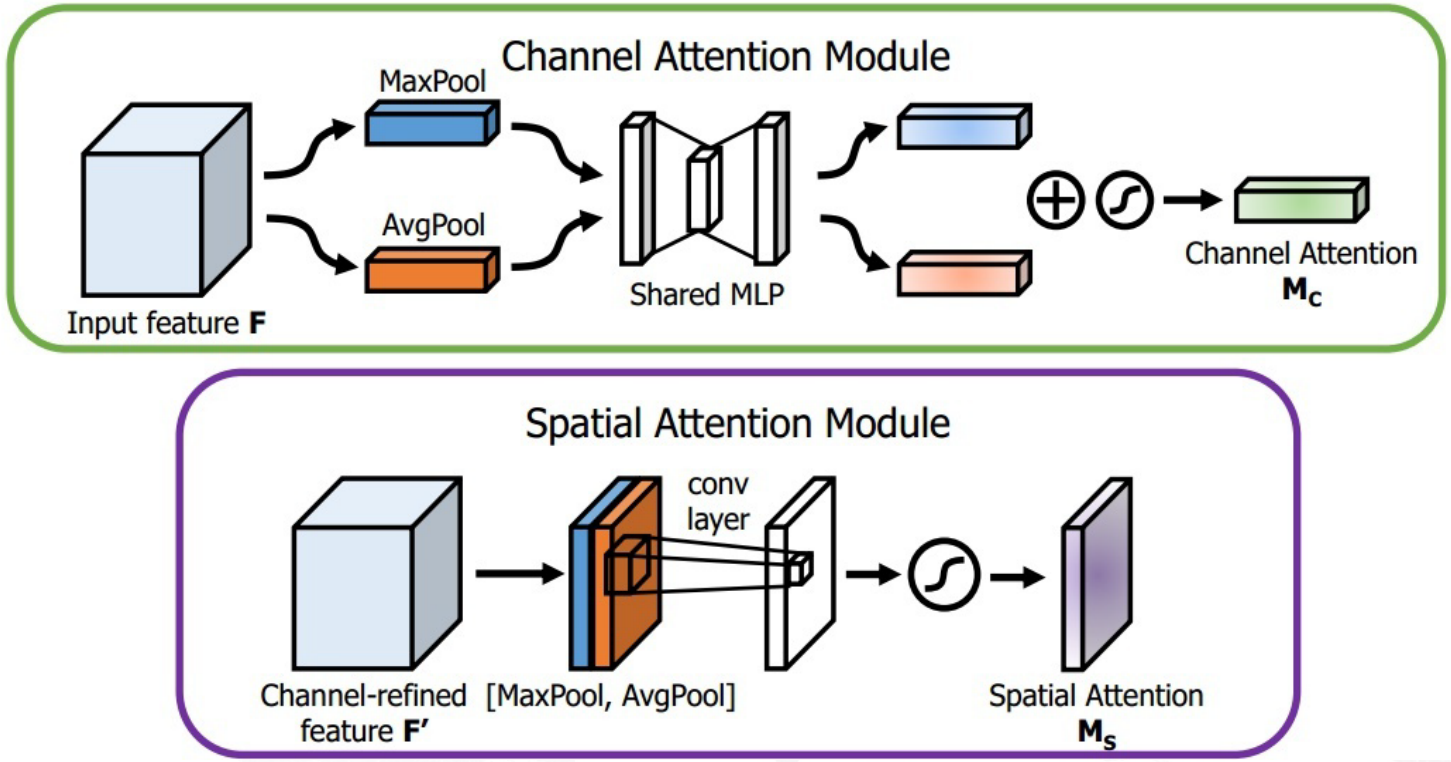


Figure 4: **Left part is an example showing the prediction results of different models.** From left to right: input frames; ground truth; FC-LSTM; 1-layer; 2-layer; 3-layer; 2-layer without convolution; 3-layer without convolution. **Right part is an example showing an "out-of-domain" run.** From left to right: input frames; ground truth; 3-layer. All sequences are sampled with an interval of 2.

02

Stand-Alone Self-Attention in Vision Models

卷积捕获长距离交互能力比较差，这是因为卷积的感受野大时缩放特性弱。此前，针对这个问题的处理方法就是用注意力机制。在卷积中运用的注意力机制主要有两种。一个是，基于通道的注意力机制；一个是，基于空间的注意力机制。这些方法有个特点，就是用全局注意力层作为卷积的附加模块；还有一个限制，因为关注输入的所有位置，要求输入比较小，否则计算成本太大。



02

Stand-Alone Self-Attention in Vision Models

The count of parameters in CNN

$$K_W * K_H * N_F * N_I$$

The count of parameters in SASA

$$1 * 1 * N_F * N_I * 3$$

The computational cost of atte ntion also grows slower with spatial extent compared to convolution with typical values of *din* and *dout*. For e xample, if *din* = *dout* = 128, a convo lution layer with *k* = 3 has the same computational cost as an attention l ayer with *k* = 19.

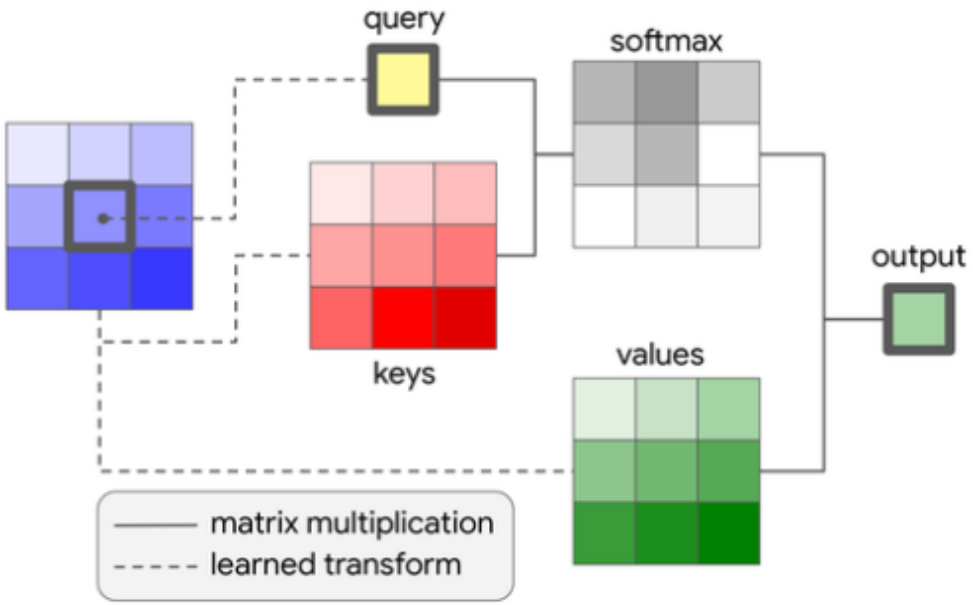


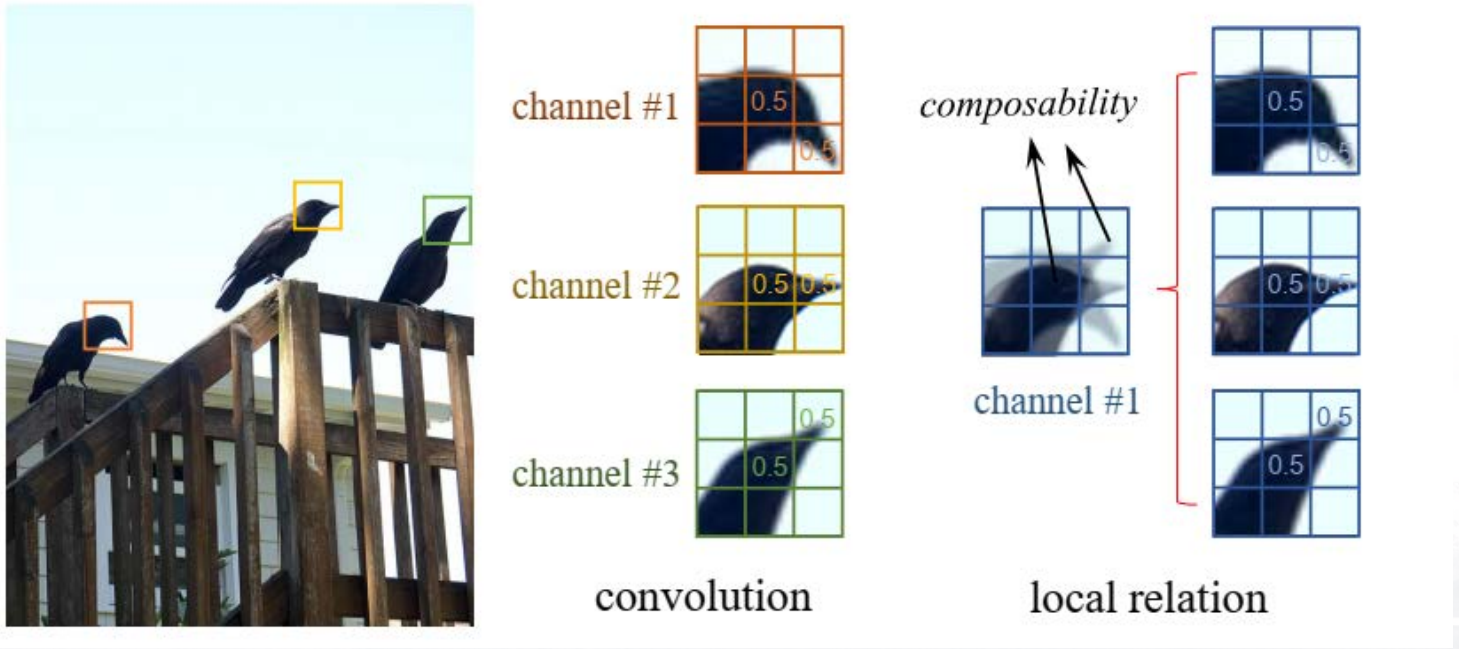
Figure 3: An example of a local attention layer over spatial extent of $k = 3$.

-1, -1	-1, 0	-1, 1	-1, 2
0, -1	0, 0	0, 1	0, 2
1, -1	1, 0	1, 1	1, 2
2, -1	2, 0	2, 1	2, 2

Figure 4: An example of relative distance computation. The relative distances are computed with respect to the position of the highlighted pixel. The format of distances is *row offset*, *column offset*.

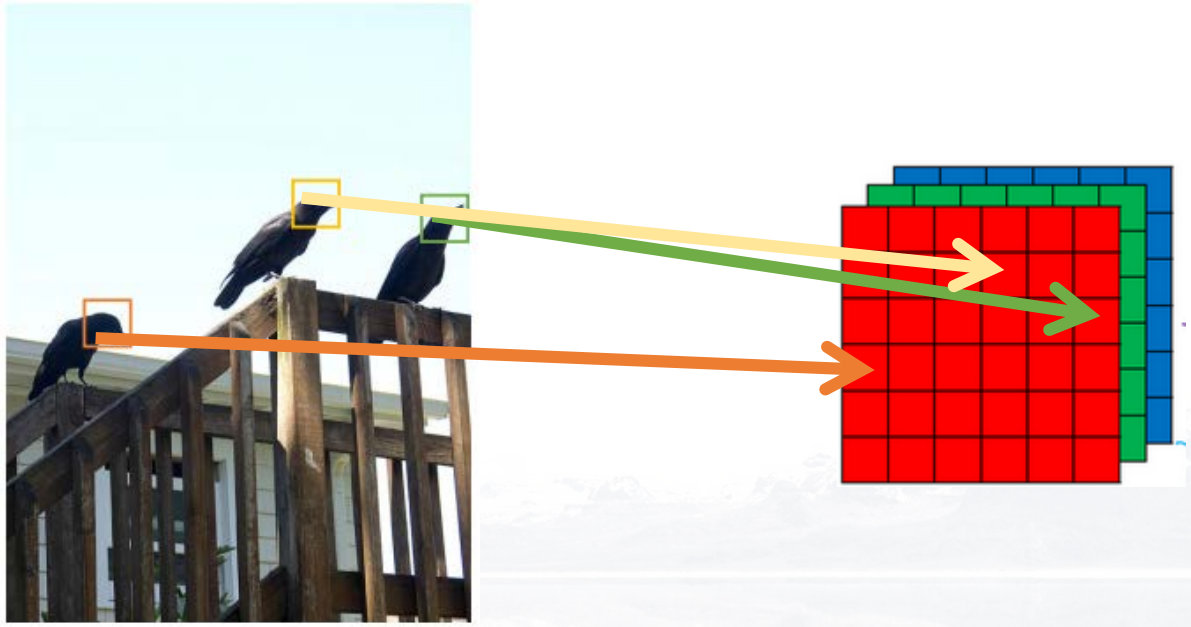
02

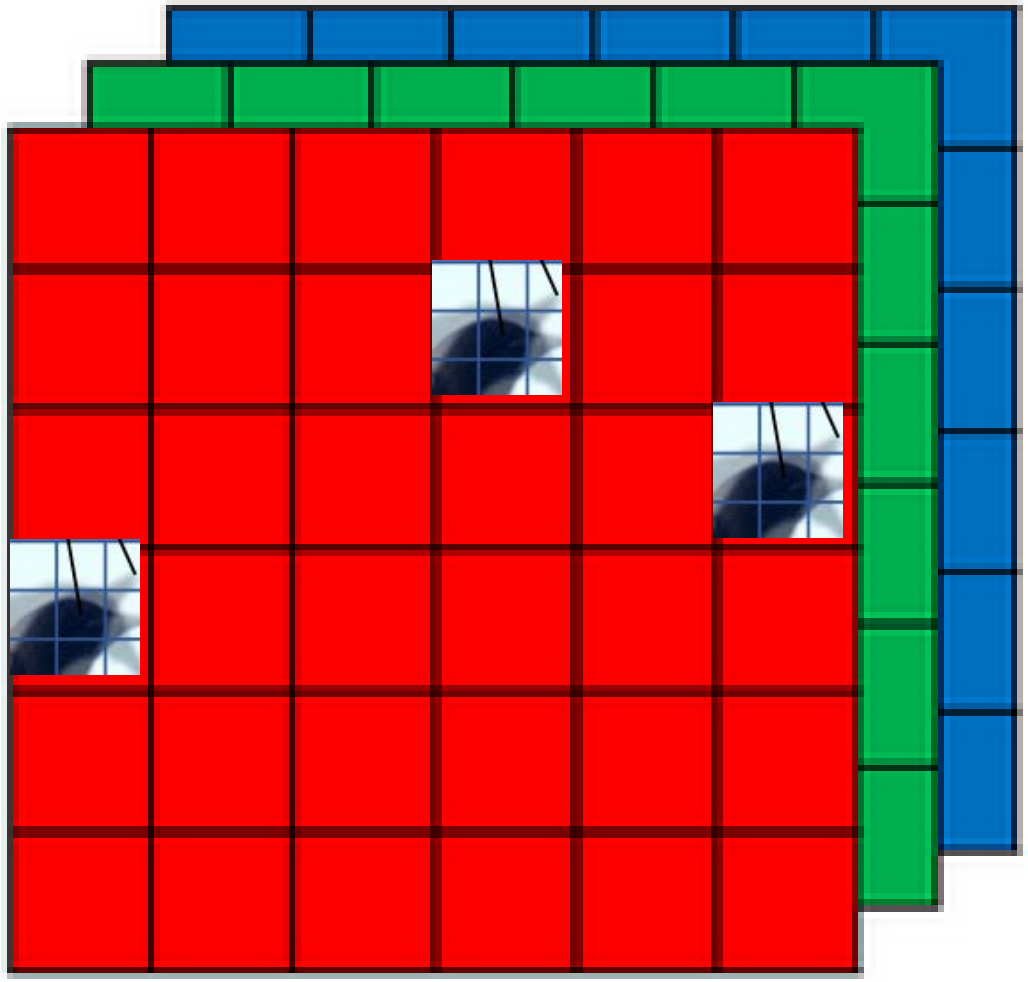
Local Relation Networks for Image Recognition



02

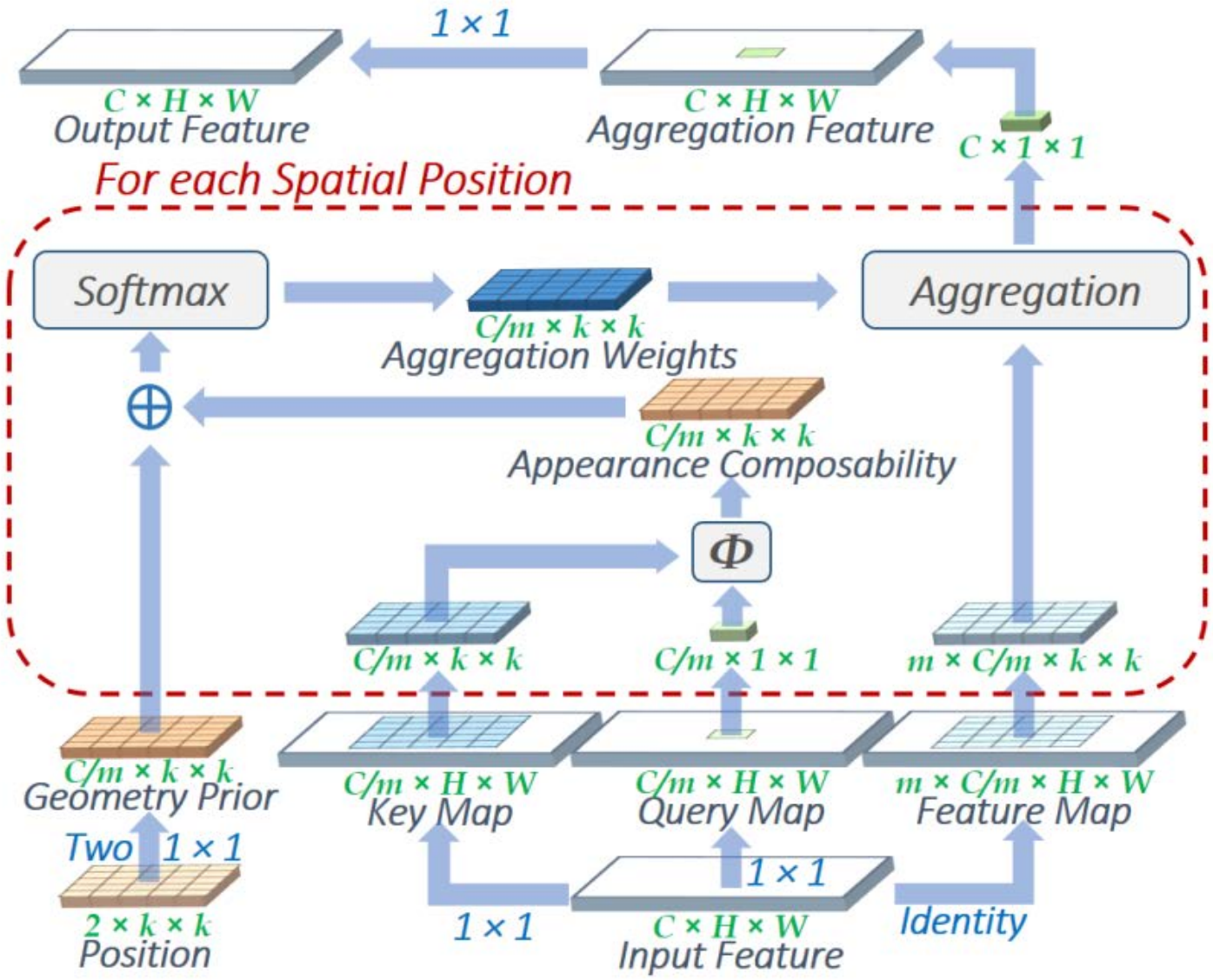
Local Relation Networks for Image Recognition





02

Local Relation Networks for Image Recognition



感谢您的观看

END