# Recurrent Space-time Graph Neural Networks

Andrei Nicolicioiu*, Iulia Duta*, Marius Leordeanu
Bitdefender, Romania

summarized by **Seongho Choi**
Seoul National University

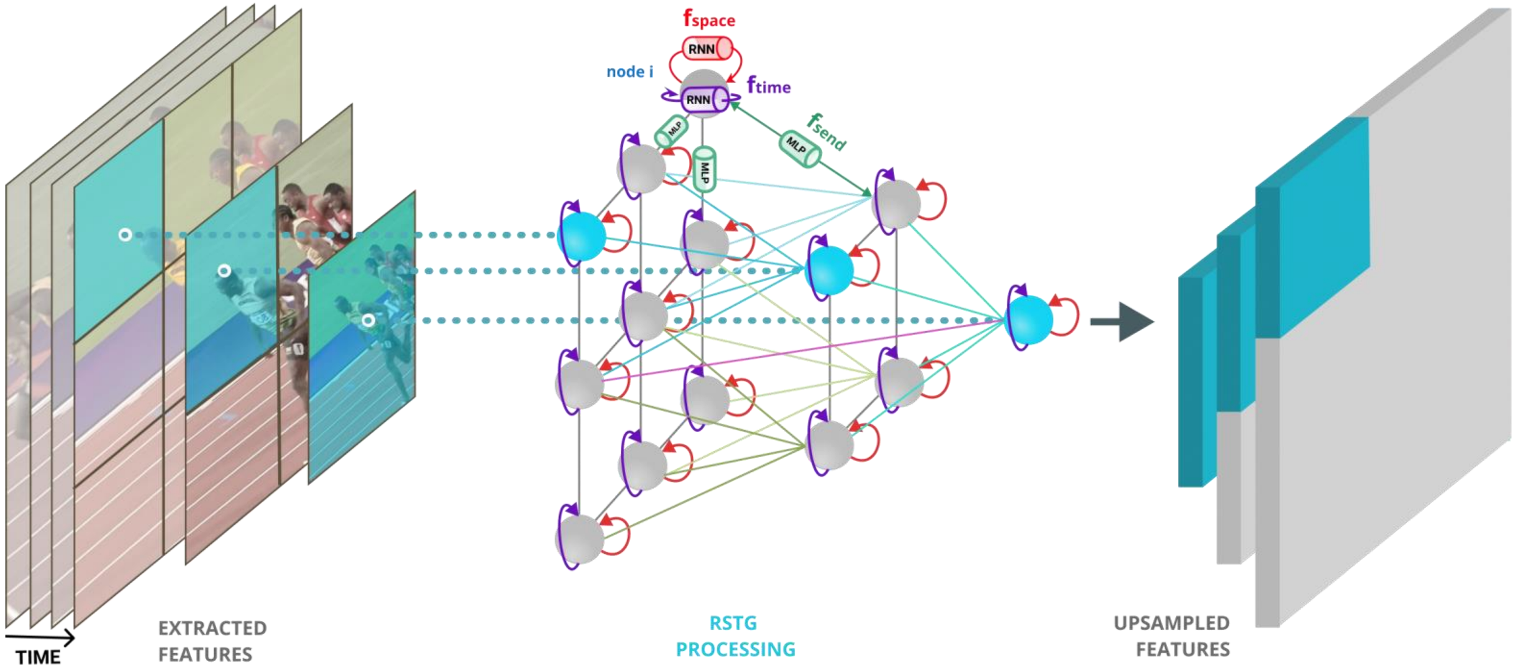
*Equal contribution

# Understanding video

- **spatial** interactions happening at the frame level
- **temporal** interactions between over time
- **long-range** interactions between distant frames

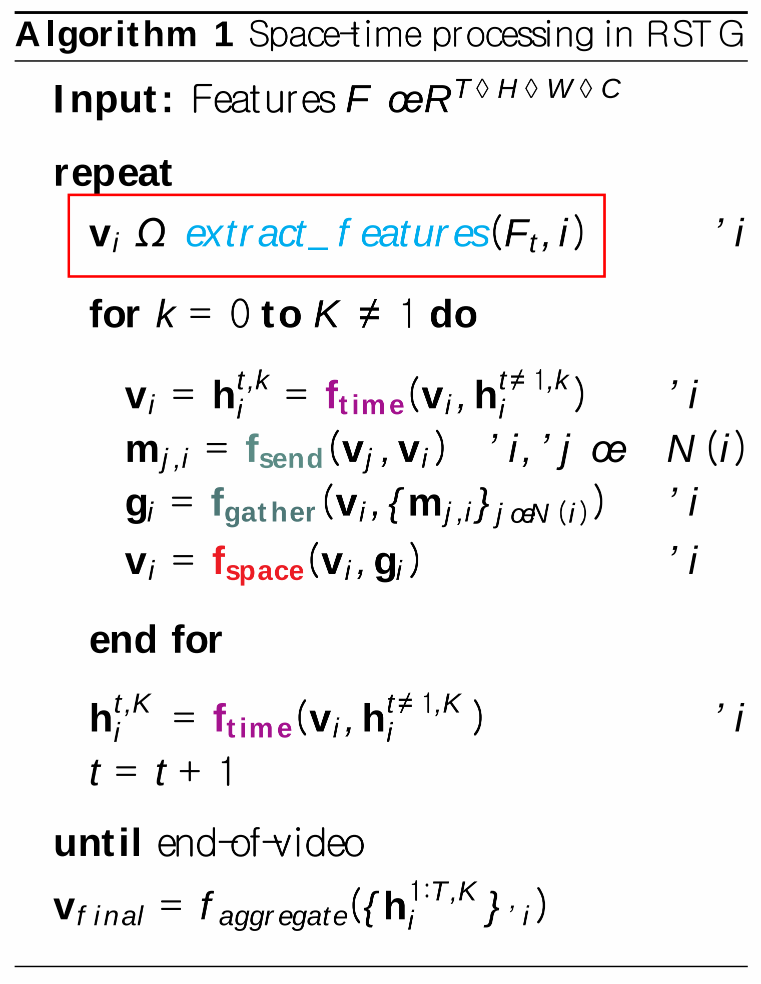
# Analysing videos with spatio-temporal graph models

- Recurrent Graph Nets are suited for video analysis tasks heavily relying on **interactions**.



```
Algorithm 1 Space-time processing in RSTG
    Input: Features $F œR^{T◊H◊W◊C}$
    repeat
        $v_i Ω extract\_features(F_t, i)$                       $'i$
        for $k = 0$ to $K ≠ 1$ do
            $v_i = h_i^{t,k} = f_{time}(v_i, h_i^{t≠1,k})$         $'i$
            $m_{j,i} = f_{send}(v_j, v_i)$      $'i, 'j œ N(i)$
            $g_i = f_{gather}(v_i, \{m_{j,i}\}_{j œN(i)})$         $'i$
            $v_i = f_{space}(v_i, g_i)$                        $'i$
        end for
        $h_i^{t,K} = f_{time}(v_i, h_i^{t≠1,K})$                 $'i$
        $t = t + 1$
    until end-of-video
    $v_{final} = f_{aggregate}(\{h_i^{1:T,K}\}_{'i})$
```

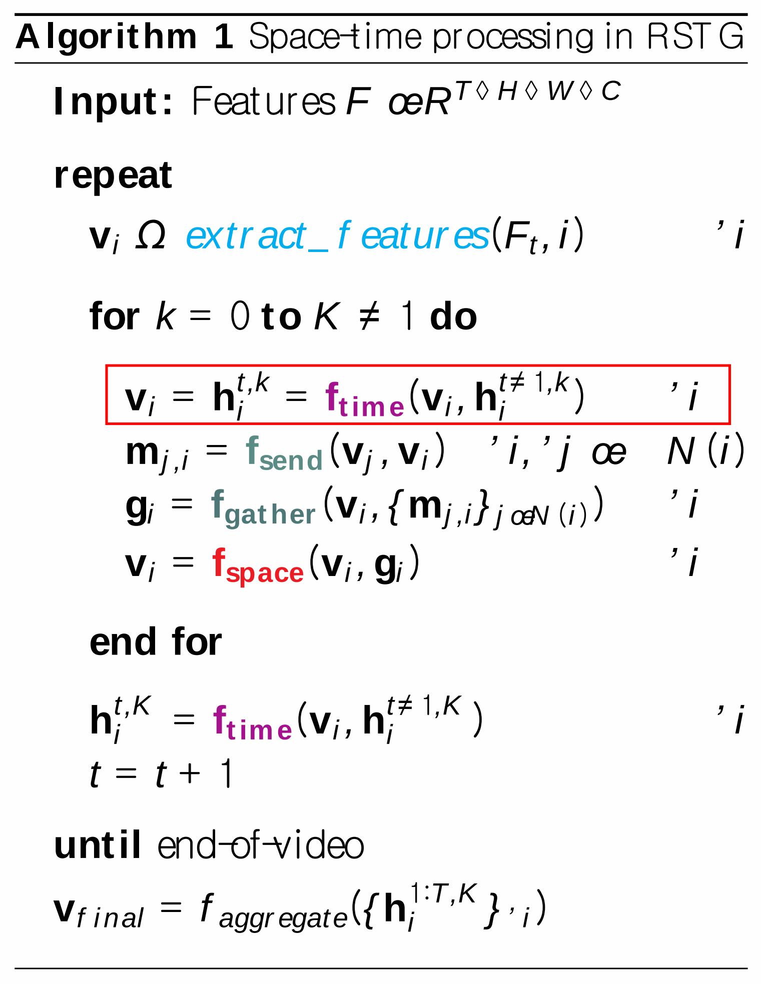
# Recurrent Space-time Graph Neural Networks



```
Algorithm 1 Space-time processing in RSTG
  Input: Features $F$ œ$R^{T◊H◊W◊C}$
  repeat
    $\mathbf{v}_i$ Ω $extract\_features(F_t, i)$        ’$i$
    for $k = 0$ to $K ≠ 1$ do
      $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,k})$        ’$i$
      $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$   ’$i$, ’$j$ œ $N(i)$
      $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j œN(i)})$        ’$i$
      $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$        ’$i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,K})$        ’$i$
    $t = t + 1$
  until end-of-video
  $\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{’i})$
```

- **Graph Creation**
  - extract features from 2D / 3D backbone
    - Mean pooling + LSTM
    - ConvNet + LSTM
    - I3D
    - Non-Local
  - arrange regions in grids at multiple scales
    - 1 × 1, 2 × 2 and 3 × 3 grids
  - each node receives information pooled from a region
  - the nodes are connected if they come from neighbouring or overlapping regions

# Recurrent Space-time Graph Neural Networks

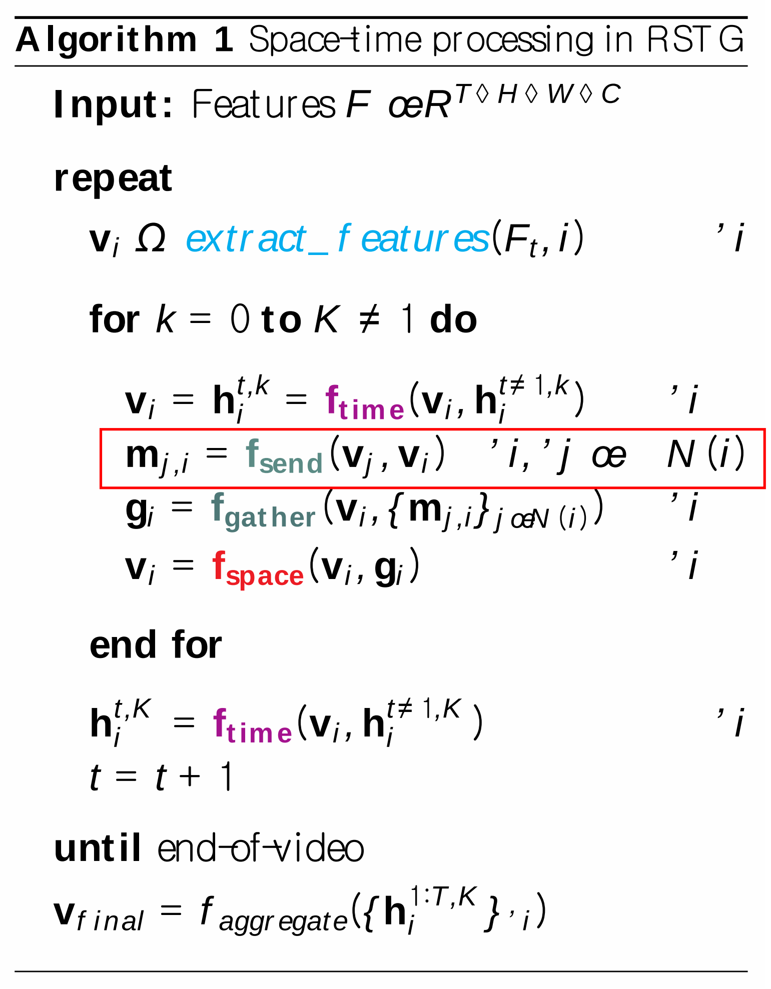


```
Algorithm 1 Space-time processing in RSTG
  Input: Features $F$ œ$R^{T◊H◊W◊C}$
  repeat
    $\mathbf{v}_i$ Ω extract_features($F_t, i$)                 ’$i$
    for $k = 0$ to $K$ ≠ 1 do
      $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,k})$        ’$i$
      $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$   ’$i$, ’$j$ œ $N(i)$
      $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j œN(i)})$      ’$i$
      $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$                  ’$i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,K})$               ’$i$
    $t = t + 1$
  until end-of-video
  $\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{’i})$
```

- **Time Processing Stage**
  - across time, each node incorporates current spatial info into the previous time step features
  - each node updates its spatial information using a recurrent function
  - no messages exchanged between different regions

$$\mathbf{h}_{i,time}^{t,k} = f_{time}(\mathbf{v}_{i,space}^{k}, \mathbf{h}_{i,time}^{t-1,k}).$$

# Recurrent Space-time Graph Neural Networks



**Algorithm 1** Space-time processing in RSTG

```
Input: Features $F\ œR^{T◊H◊W◊C}$
repeat
  $\mathbf{v}_i\ Ω$ extract_features$(F_t, i)$    ’ i
  for $k = 0$ to $K ≠ 1$ do
    $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,k})$    ’ i
    $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    ’ i, ’ j œ N(i)
    $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j\,œN(i)})$    ’ i
    $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    ’ i
  end for
  $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,K})$    ’ i
  $t = t + 1$
until end-of-video
$\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{’i})$
```

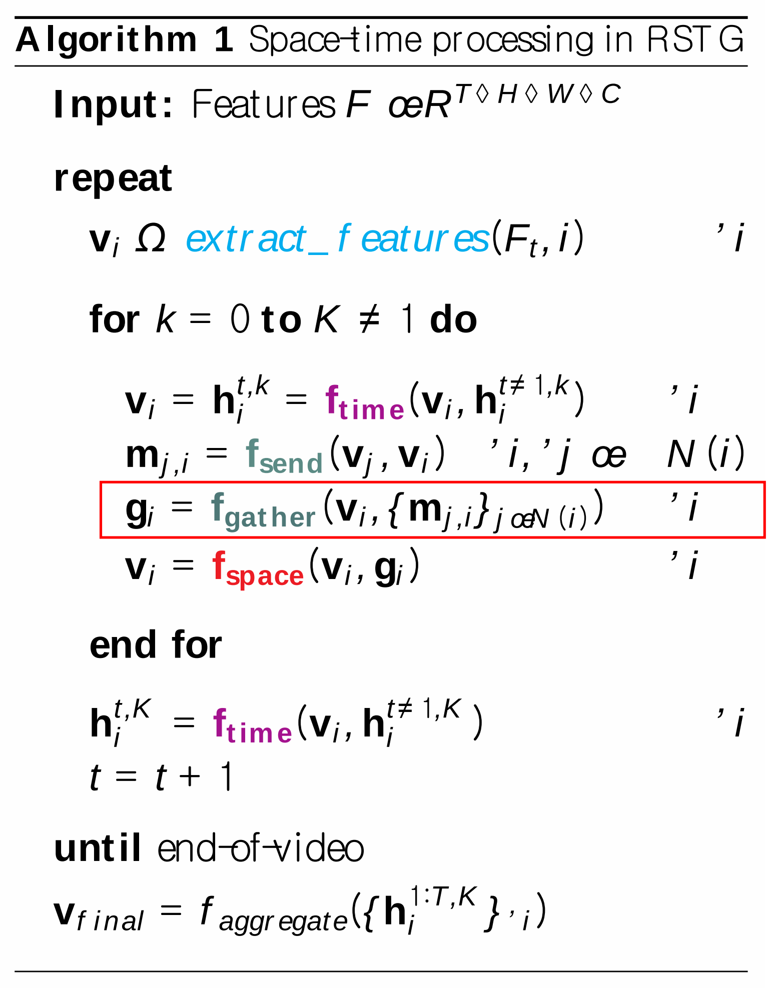
- **Space Processing Stage**
  - **Send**: messages represent pairwise spatial interactions

$$f_{send}(\mathbf{v}_j, \mathbf{v}_i) = \mathrm{MLP}_s([\mathbf{v}_j|\mathbf{v}_i]) \in \mathbb{R}^D.$$

$$\mathrm{MLP}_a(\mathbf{x}) = \sigma(W_{a_2}\sigma(W_{a_1}(\mathbf{x}) + b_{a_1}) + b_{a_2}).$$

  - Positional Awareness: Each source node should be aware of the destination node’s position. We concatenate the position of both nodes to the input of $f_{send}$

# Recurrent Space-time Graph Neural Networks



**Algorithm 1** Space-time processing in RSTG

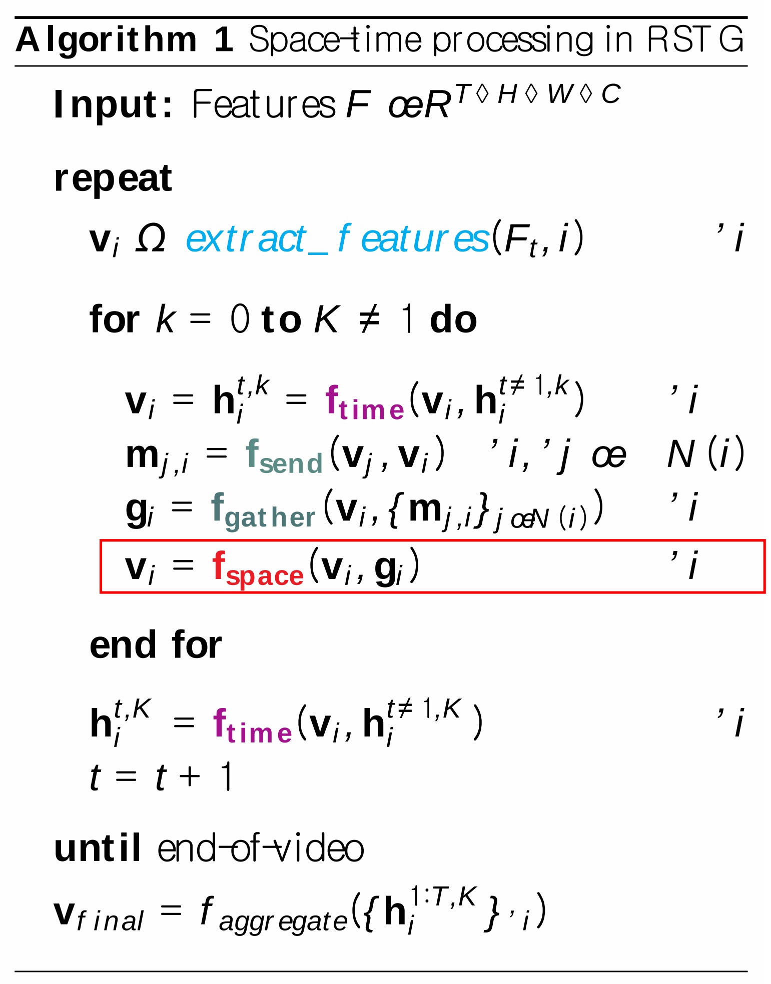
```
Input: Features $F \in R^{T \times H \times W \times C}$
repeat
    $\mathbf{v}_i \leftarrow extract\_features(F_t, i)$    $\forall i$
    for $k = 0$ to $K - 1$ do
        $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,k})$    $\forall i$
        $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    $\forall i, \forall j \in N(i)$
        $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j \in N(i)})$    $\forall i$
        $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    $\forall i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,K})$    $\forall i$
    $t = t + 1$
until end-of-video
$\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{\forall i})$
```

- **Space Processing Stage**
  - **Gather**: aggregate received messages by an attention mechanism

$$f_{gather}(\mathbf{v}_i) = \sum_{j \in \mathcal{N}(i)} \alpha(\mathbf{v}_j, \mathbf{v}_i) f_{send}(\mathbf{v}_j, \mathbf{v}_i) \in \mathbb{R}^D.$$

$$\alpha(\mathbf{v}_j, \mathbf{v}_i) = (W_{\alpha_1} \mathbf{v}_j)^T (W_{\alpha_2} \mathbf{v}_i) \in \mathbb{R}.$$

# Recurrent Space-time Graph Neural Networks

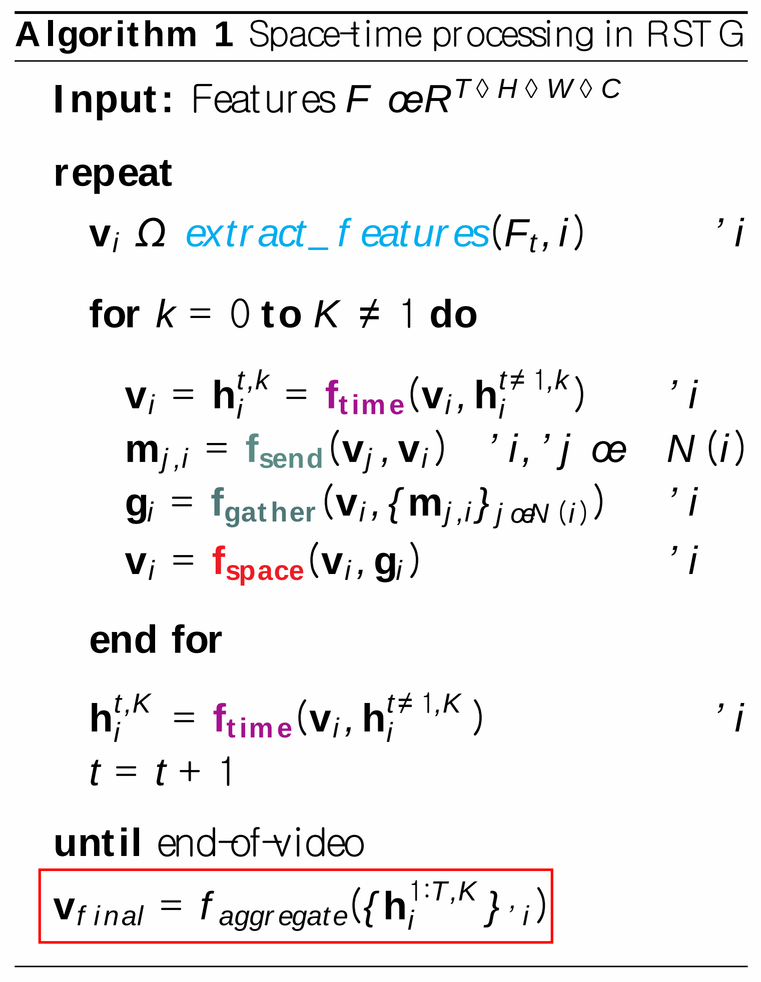


```
Algorithm 1 Space-time processing in RSTG
  Input: Features $F \in R^{T \times H \times W \times C}$
  repeat
    $\mathbf{v}_i \leftarrow extract\_features(F_t, i)$    $\forall i$
    for $k = 0$ to $K - 1$ do
      $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,k})$    $\forall i$
      $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    $\forall i, \forall j \in N(i)$
      $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j \in N(i)})$    $\forall i$
      $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    $\forall i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,K})$    $\forall i$
    $t = t + 1$
  until end-of-video
  $\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{\forall i})$
```

- **Space Processing Stage**
  - **Update**: incorporate global context into each local information

$$f_{space}(\mathbf{v}_i) = \mathrm{MLP}_u([\mathbf{v}_i | f_{gather}(\mathbf{v}_i)]) \in \mathbb{R}^D.$$

# Recurrent Space-time Graph Neural Networks

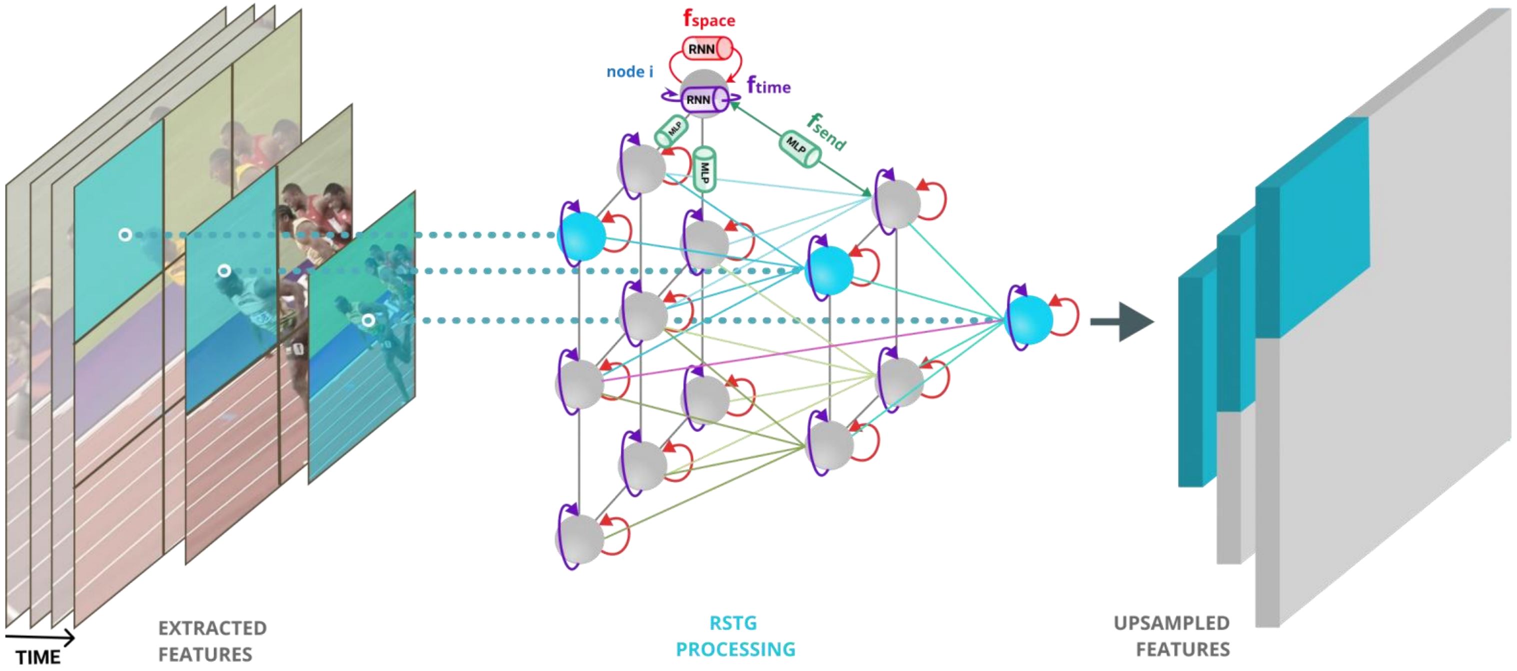


```
Algorithm 1 Space-time processing in RSTG
  Input: Features $F \in R^{T \times H \times W \times C}$
  repeat
    $\mathbf{v}_i \leftarrow extract\_features(F_t, i)$    $\forall i$
    for $k = 0$ to $K - 1$ do
      $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,k})$    $\forall i$
      $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    $\forall i, \forall j \in N(i)$
      $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j \in N(i)})$    $\forall i$
      $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    $\forall i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,K})$    $\forall i$
    $t = t + 1$
  until end-of-video
  $\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{\forall i})$
```

- **Aggregation step (versatile usage)**
  - **RSTG-to-vec**: obtain **1D vector** by summing the all the nodes from the last time step
  - **RSTG-to-map**: obtain f**eatures volume** with the same size as the input, by projecting back each node into initial corresponding region

# Analysing videos with spatio-temporal graph models

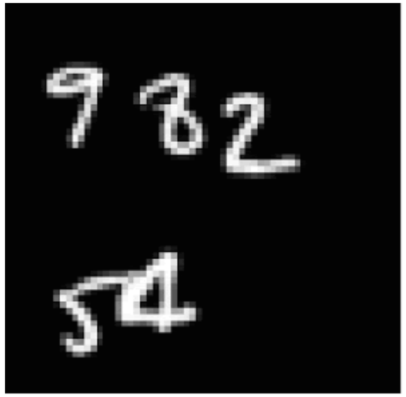
- Recurrent Graph Nets are suited for video analysis tasks heavily relying on **interactions**.



```
Algorithm 1 Space-time processing in RSTG
  Input: Features $F œR^{T◊H◊W◊C}$
  repeat
    $\mathbf{v}_i$ Ω extract_features($F_t, i$)    ’i
    for $k = 0$ to $K ≠ 1$ do
      $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,k})$    ’i
      $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    ’i, ’j œ N(i)
      $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j œN(i)})$    ’i
      $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    ’i
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,K})$    ’i
    $t = t + 1$
  until end-of-video
  $\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{’i})$
```

# SyncMNIST dataset (proposed novel dataset)

- SyncMNIST
  - the complexity comes from modeling spatial and temporal interactions
  - cleaner, simpler environment.



**Sync pair - (4,2)**

**Random**

# SyncMNIST dataset (proposed novel dataset)

Table 1: Accuracy on SyncMNIST dataset, showing the capabilities of different parts of our model.

| Model | 3 SyncMNIST | 5 SyncMNIST |
|---|---|---|
| Mean + LSTM | 77.0 | – |
| Conv + LSTM | 95.0 | 39.7 |
| I3D | – | 90.6 |
| Non-Local | – | 93.5 |
| RSTG: Space-Only | 61.3 | – |
| RSTG: Time-Only | 89.7 | – |
| RSTG: Homogenous | 95.7 | 58.3 |
| RSTG: 1-temp-stage | 97.0 | 74.1 |
| RSTG: All-temp-stages | 98.9 | 94.5 |
| RSTG: Positional All-temp | – | 97.2 |

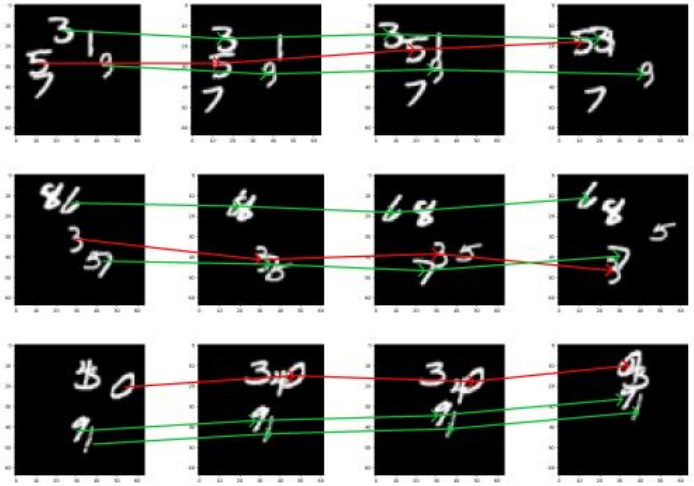


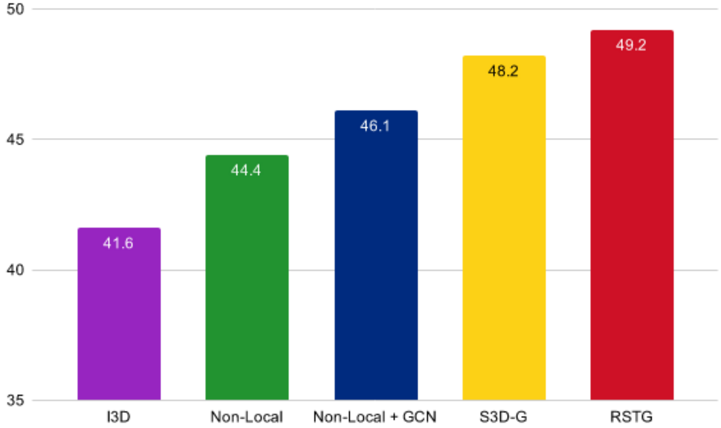
Figure 3: On each row we present frames from videos of 5SyncMNIST dataset. In each video sequence two digits follow the exact same pattern of movement. The correct classes: "3-9" "6-7" and "9-1".

# Real world experiments

- Something-Something v1
    - human-object interaction dataset
    - interactions between entities across the entire video are essential

a. Failing to put smt into smt because smt does not fit

b. Pretend to put smt into smt



# Real world experiments

- RSTG shows state of the art performance

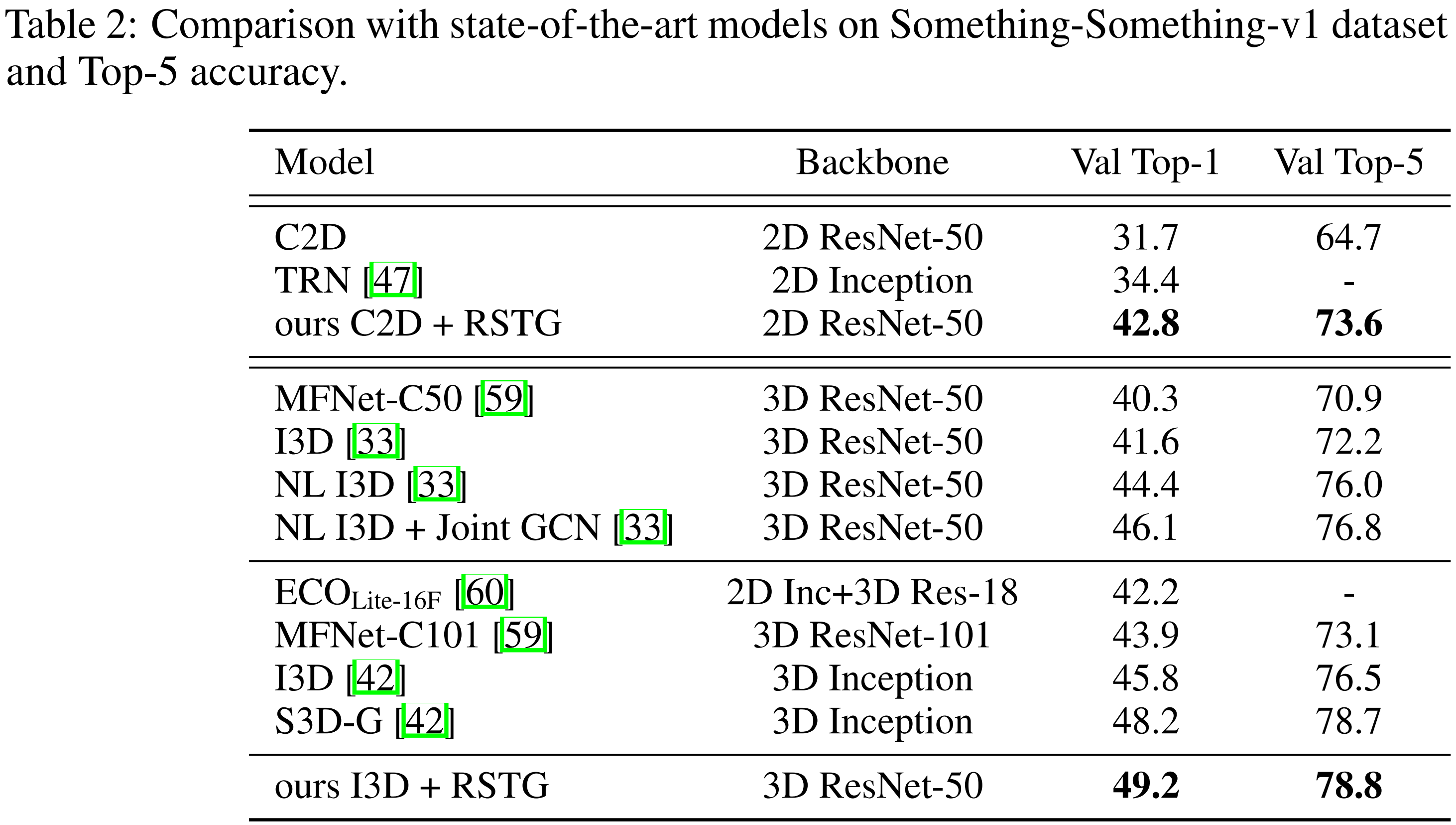


Table 2: Comparison with state-of-the-art models on Something-Something-v1 dataset showing Top-1 and Top-5 accuracy.

| Model | Backbone | Val Top-1 | Val Top-5 |
|---|---|---|---|
| C2D | 2D ResNet-50 | 31.7 | 64.7 |
| TRN [47] | 2D Inception | 34.4 | - |
| ours C2D + RSTG | 2D ResNet-50 | **42.8** | **73.6** |
| MFNet-C50 [59] | 3D ResNet-50 | 40.3 | 70.9 |
| I3D [33] | 3D ResNet-50 | 41.6 | 72.2 |
| NL I3D [33] | 3D ResNet-50 | 44.4 | 76.0 |
| NL I3D + Joint GCN [33] | 3D ResNet-50 | 46.1 | 76.8 |
| ECO$_{\text{Lite-16F}}$ [60] | 2D Inc+3D Res-18 | 42.2 | - |
| MFNet-C101 [59] | 3D ResNet-101 | 43.9 | 73.1 |
| I3D [42] | 3D Inception | 45.8 | 76.5 |
| S3D-G [42] | 3D Inception | 48.2 | 78.7 |
| ours I3D + RSTG | 3D ResNet-50 | **49.2** | **78.8** |

# Summary & Limitation

- RSTG (Recurrent Space-time Graph Neural Networks)
  - propose a general neural graph block for learning in spatio-temporal domain
  - factorize space and time and process them differently
  - achieves a relatively low computational complexity
  - introduce a synthetic dataset involving explicit space-time interactions
  - shows state-of-the-art performance on real world dataset

- Limitation
  - not various experiments on other dataset
  - not modeling long-range interactions

# Installation Guide

- **Install Conda**
  - https://www.anaconda.com/products/individual
  - For Linux, 64-Bit (x86) Installer (522 MB)
  - `bash Anaconda-latest-Linux-x86_64.sh`

- **Create the new environment with python 3.6**
  - `conda create -n py36 python=3.6 anaconda`
  - `conda activate py36`

- **Install tensorflow-gpu 1.13.1**
  - `conda install -c anaconda tensorflow-gpu==1.13.1`

- **Check tensorflow version**
  - `python -c 'import tensorflow as tf; print(tf.__version__)'`

# Installation Guide

- **Git clone**
  - git clone https://github.com/IuliaDuta/RSTG.git
- **Download checkpoints for pre-trained models**
  - You can find here some checkpoint and please put the checkpoints in ./checkpoints/
  - Download “model_backbone_i3d_rstg_res3_res4”
- **Download Something-Something Dataset (need 26GB space)**
  - Download Something-Something-v1 dataset and extract it in ./datasets/

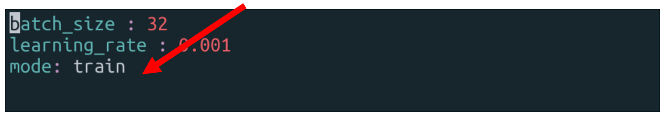
# Dataset preprocessing

- **Something-Something Dataset (need 14.7GB space)**
  - split train/valid/test with the code in this slide note (it's not included in github source code)
  - ./scripts/create_smt-smt_dataset_train.sh
  - ./scripts/create_smt-smt_dataset_valid.sh

- **SyncMNIST Dataset (need 94.6GB space)**
  - ./scripts/create_syncMNIST.sh
  - ./scripts/create_syncMNIST_test.sh

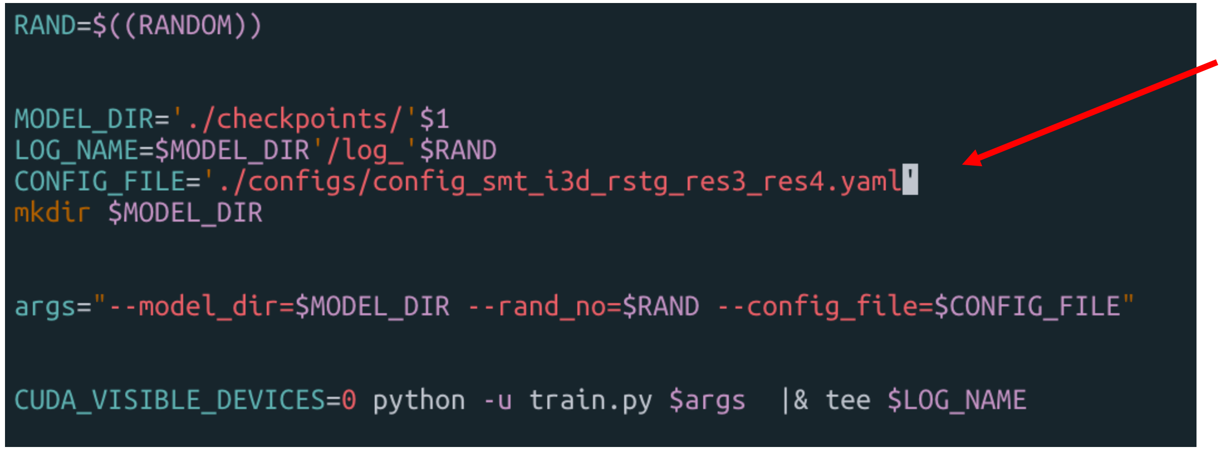
# How to evaluate

- **Something-Something Dataset**
  - set the flag --mode=train (in config file)



```
batch_size : 32
learning_rate : 0.001
mode: train
```

  - ./scripts/run_smt-smt.sh model name

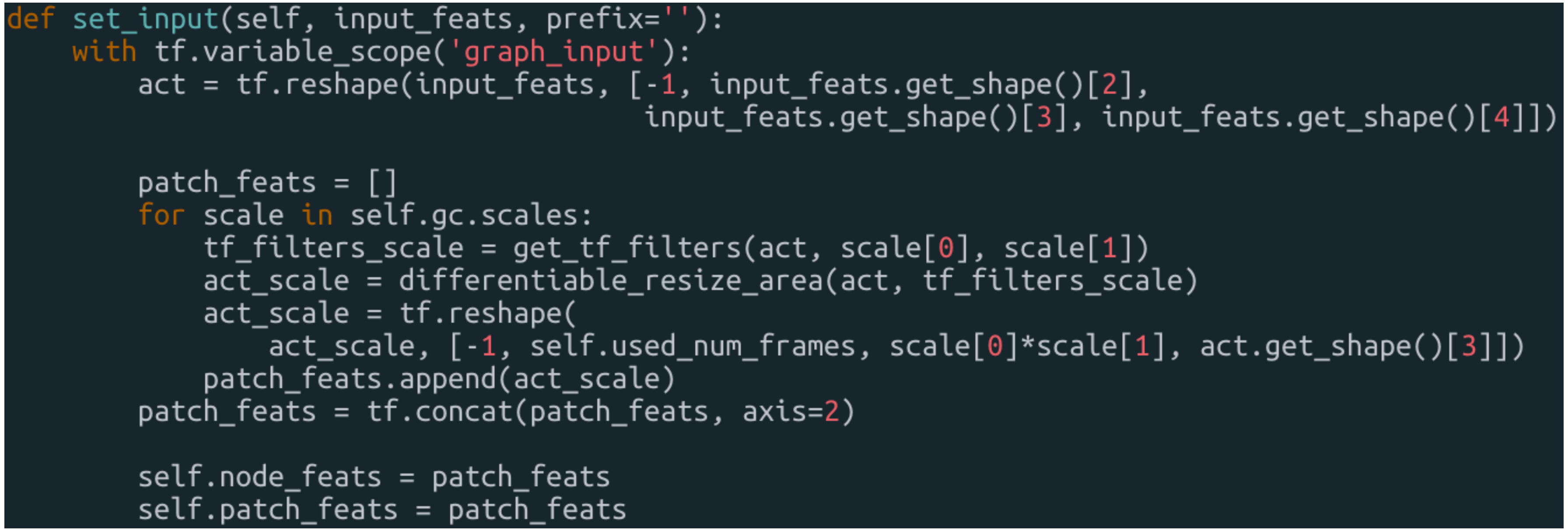


```
RAND=$((RANDOM))

MODEL_DIR='./checkpoints/'$1
LOG_NAME=$MODEL_DIR'/log_'$RAND
CONFIG_FILE='./configs/config_smt_i3d_rstg_res3_res4.yaml'
mkdir $MODEL_DIR

args="--model_dir=$MODEL_DIR --rand_no=$RAND --config_file=$CONFIG_FILE"

CUDA_VISIBLE_DEVICES=0 python -u train.py $args  |& tee $LOG_NAME
```

give the path to the config file

# Code Explanation (graph_model/graph_model.py)

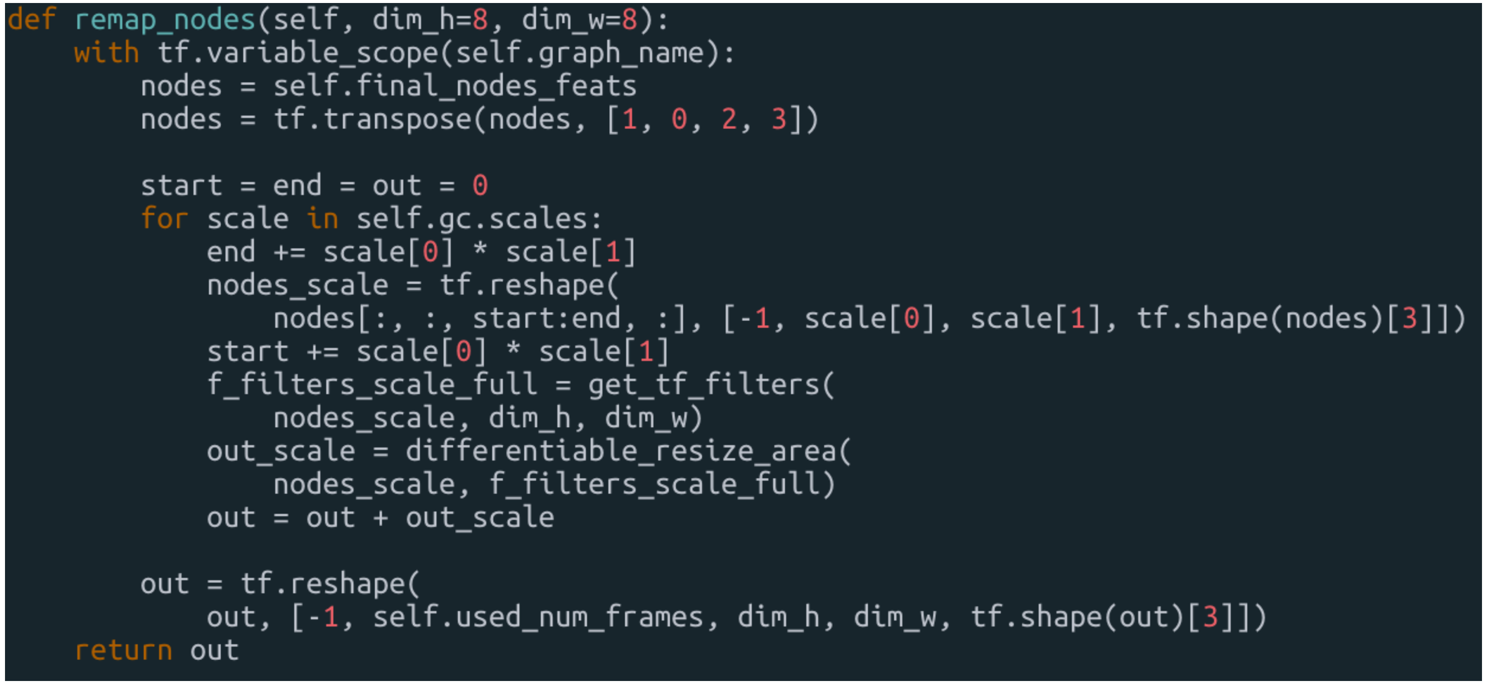
- **set_input**
  - create the nodes from the backbone's feature maps
  - arrange regions in grids at multiple scales



```
def set_input(self, input_feats, prefix=''):
    with tf.variable_scope('graph_input'):
        act = tf.reshape(input_feats, [-1, input_feats.get_shape()[2],
                                       input_feats.get_shape()[3], input_feats.get_shape()[4]])

        patch_feats = []
        for scale in self.gc.scales:
            tf_filters_scale = get_tf_filters(act, scale[0], scale[1])
            act_scale = differentiable_resize_area(act, tf_filters_scale)
            act_scale = tf.reshape(
                act_scale, [-1, self.used_num_frames, scale[0]*scale[1], act.get_shape()[3]])
            patch_feats.append(act_scale)
        patch_feats = tf.concat(patch_feats, axis=2)

        self.node_feats = patch_feats
        self.patch_feats = patch_feats
```

# Code Explanation (graph_model/graph_model.py)

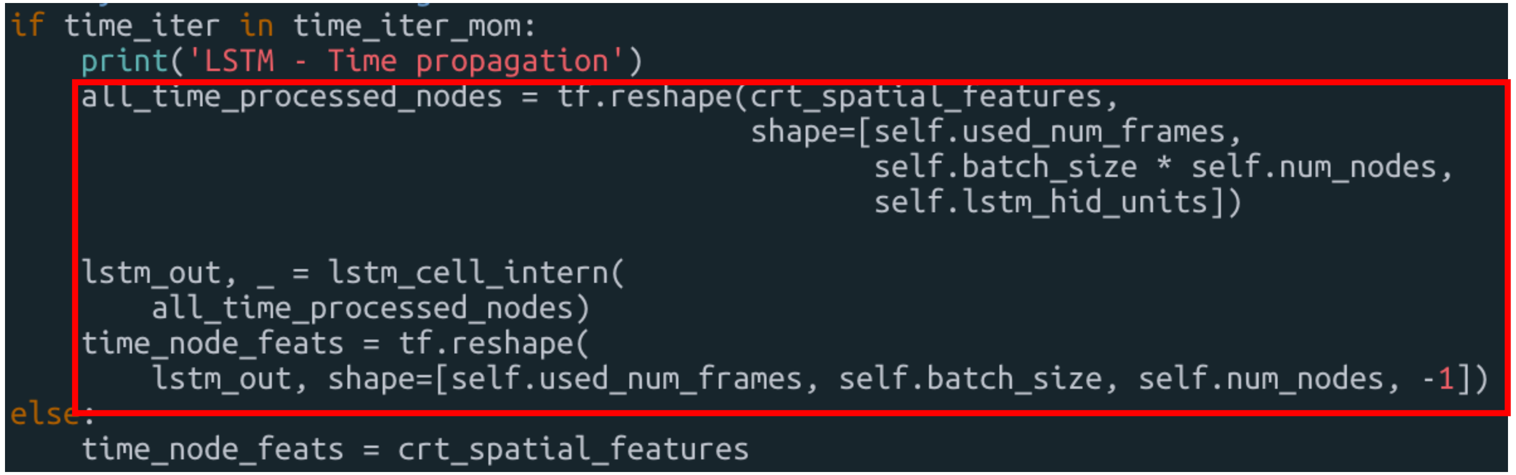
- **remap_nodes**
  - project features from graph to features map
  - obtain **features volume** with the same size as the input, by projecting back each node into initial corresponding region



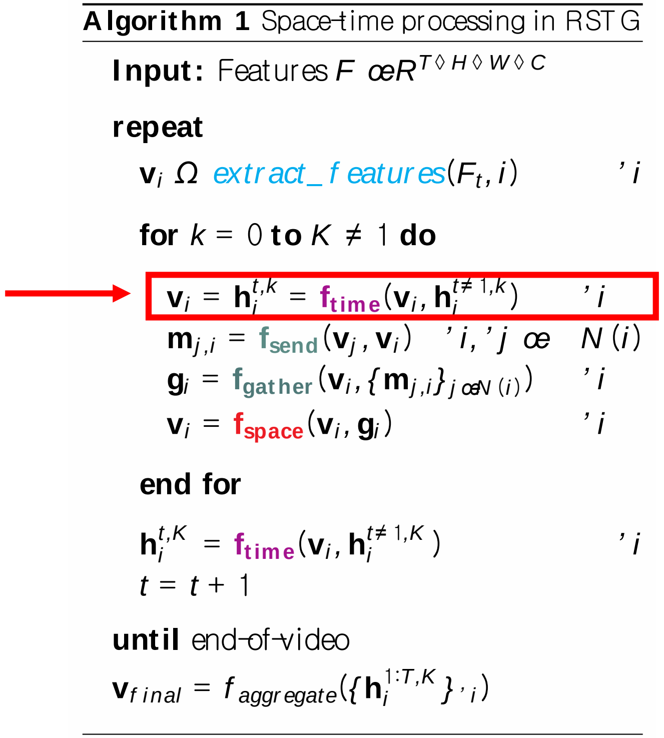
```
def remap_nodes(self, dim_h=8, dim_w=8):
    with tf.variable_scope(self.graph_name):
        nodes = self.final_nodes_feats
        nodes = tf.transpose(nodes, [1, 0, 2, 3])

        start = end = out = 0
        for scale in self.gc.scales:
            end += scale[0] * scale[1]
            nodes_scale = tf.reshape(
                nodes[:, :, start:end, :], [-1, scale[0], scale[1], tf.shape(nodes)[3]])
            start += scale[0] * scale[1]
            f_filters_scale_full = get_tf_filters(
                nodes_scale, dim_h, dim_w)
            out_scale = differentiable_resize_area(
                nodes_scale, f_filters_scale_full)
            out = out + out_scale

        out = tf.reshape(
            out, [-1, self.used_num_frames, dim_h, dim_w, tf.shape(out)[3]])
    return out
```

# Code Explanation (graph_model/graph_model.py)

- **Time Processing Stage**
  - each node updates its spatial information using a recurrent function
  - no messages exchanged between different regions



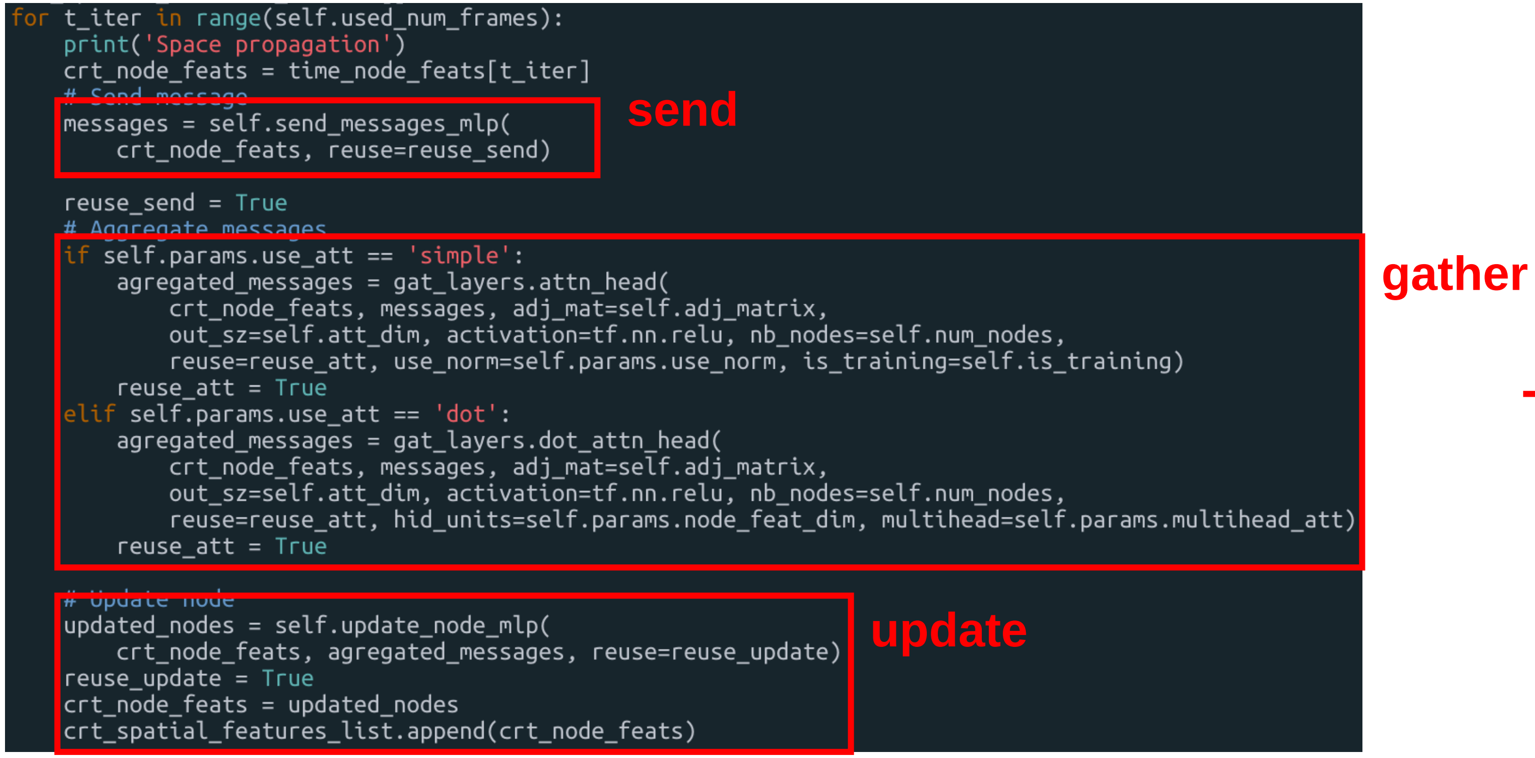
```python
if time_iter in time_iter_mom:
    print('LSTM - Time propagation')
    all_time_processed_nodes = tf.reshape(crt_spatial_features,
                                          shape=[self.used_num_frames,
                                                 self.batch_size * self.num_nodes,
                                                 self.lstm_hid_units])

    lstm_out, _ = lstm_cell_intern(
        all_time_processed_nodes)
    time_node_feats = tf.reshape(
        lstm_out, shape=[self.used_num_frames, self.batch_size, self.num_nodes, -1])
else:
    time_node_feats = crt_spatial_features
```



```
Algorithm 1 Space-time processing in RSTG
Input: Features $F \in R^{T\times H\times W\times C}$
repeat
    $\mathbf{v}_i \leftarrow extract\_features(F_t, i)$   $\forall i$
    for $k = 0$ to $K-1$ do
        $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,k})$   $\forall i$
        $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$   $\forall i, \forall j \in N(i)$
        $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j\in N(i)})$   $\forall i$
        $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$   $\forall i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,K})$   $\forall i$
    $t = t + 1$
until end-of-video
$\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{\forall i})$
```

# Code Explanation (graph_model/graph_model.py)

## ▪ Space Processing Stage



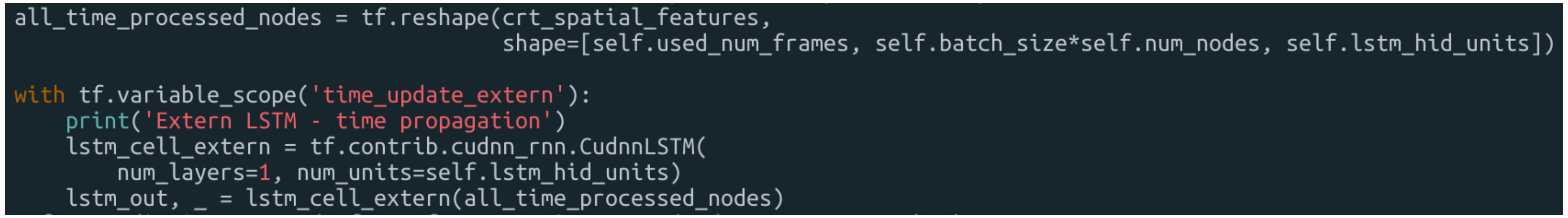
```
for t_iter in range(self.used_num_frames):
    print('Space propagation')
    crt_node_feats = time_node_feats[t_iter]
    # Send message
    messages = self.send_messages_mlp(
        crt_node_feats, reuse=reuse_send)

    reuse_send = True
    # Aggregate messages
    if self.params.use_att == 'simple':
        agregated_messages = gat_layers.attn_head(
            crt_node_feats, messages, adj_mat=self.adj_matrix,
            out_sz=self.att_dim, activation=tf.nn.relu, nb_nodes=self.num_nodes,
            reuse=reuse_att, use_norm=self.params.use_norm, is_training=self.is_training)
        reuse_att = True
    elif self.params.use_att == 'dot':
        agregated_messages = gat_layers.dot_attn_head(
            crt_node_feats, messages, adj_mat=self.adj_matrix,
            out_sz=self.att_dim, activation=tf.nn.relu, nb_nodes=self.num_nodes,
            reuse=reuse_att, hid_units=self.params.node_feat_dim, multihead=self.params.multihead_att)
        reuse_att = True

    # Update node
    updated_nodes = self.update_node_mlp(
        crt_node_feats, agregated_messages, reuse=reuse_update)
    reuse_update = True
    crt_node_feats = updated_nodes
    crt_spatial_features_list.append(crt_node_feats)
```

**Algorithm 1** Space-time processing in RSTG

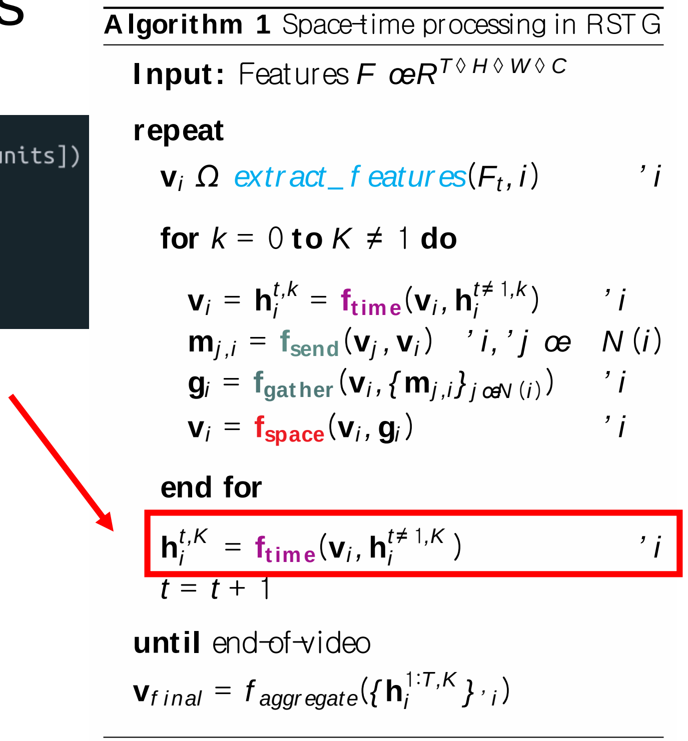
```
Input: Features $F \in R^{T \times H \times W \times C}$
repeat
    $\mathbf{v}_i \leftarrow extract\_features(F_t, i)$    $\forall i$
    for $k = 0$ to $K - 1$ do
        $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,k})$    $\forall i$
        $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    $\forall i, \forall j \in N(i)$
        $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j \in N(i)})$    $\forall i$
        $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    $\forall i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t-1,K})$    $\forall i$
    $t = t + 1$
until end-of-video
$\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{\forall i})$
```

# Code Explanation (graph_model/graph_model.py)

- **Time Processing Stage**
  - each node updates its spatial information using a recurrent function
  - no messages exchanged between different regions



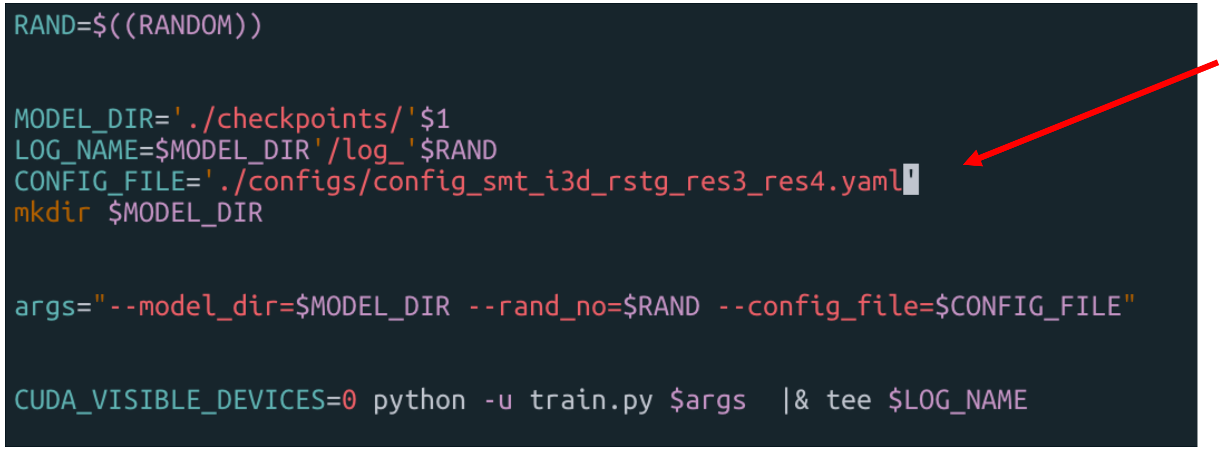
```
all_time_processed_nodes = tf.reshape(crt_spatial_features,
                                      shape=[self.used_num_frames, self.batch_size*self.num_nodes, self.lstm_hid_units])

with tf.variable_scope('time_update_extern'):
    print('Extern LSTM - time propagation')
    lstm_cell_extern = tf.contrib.cudnn_rnn.CudnnLSTM(
        num_layers=1, num_units=self.lstm_hid_units)
    lstm_out, _ = lstm_cell_extern(all_time_processed_nodes)
```



```
Algorithm 1 Space-time processing in RSTG
  Input: Features $F$ œ$R^{T◊H◊W◊C}$
  repeat
    $\mathbf{v}_i$ Ω $extract\_features(F_t, i)$    ’$i$
    for $k = 0$ to $K ≠ 1$ do
      $\mathbf{v}_i = \mathbf{h}_i^{t,k} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,k})$    ’$i$
      $\mathbf{m}_{j,i} = \mathbf{f}_{send}(\mathbf{v}_j, \mathbf{v}_i)$    ’$i$, ’$j$ œ $N(i)$
      $\mathbf{g}_i = \mathbf{f}_{gather}(\mathbf{v}_i, \{\mathbf{m}_{j,i}\}_{j œN(i)})$    ’$i$
      $\mathbf{v}_i = \mathbf{f}_{space}(\mathbf{v}_i, \mathbf{g}_i)$    ’$i$
    end for
    $\mathbf{h}_i^{t,K} = \mathbf{f}_{time}(\mathbf{v}_i, \mathbf{h}_i^{t≠1,K})$    ’$i$
    $t = t + 1$
  until end-of-video
  $\mathbf{v}_{final} = f_{aggregate}(\{\mathbf{h}_i^{1:T,K}\}_{’i})$
```

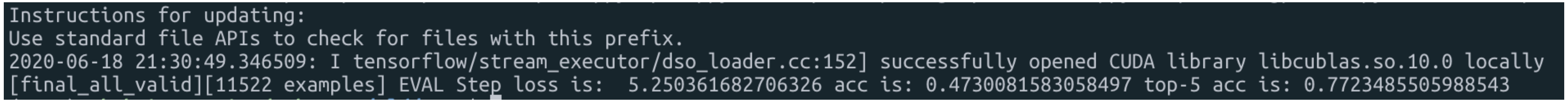
# How to evaluate

- **Something-Something Dataset**
  - ./scripts/run_smt-smt.sh



```
RAND=$((RANDOM))

MODEL_DIR='./checkpoints/'$1
LOG_NAME=$MODEL_DIR'/log_'$RAND
CONFIG_FILE='./configs/config_smt_i3d_rstg_res3_res4.yaml'
mkdir $MODEL_DIR

args="--model_dir=$MODEL_DIR --rand_no=$RAND --config_file=$CONFIG_FILE"

CUDA_VISIBLE_DEVICES=0 python -u train.py $args  |& tee $LOG_NAME
```

give the path to the config file



```
Instructions for updating:
Use standard file APIs to check for files with this prefix.
2020-06-18 21:30:49.346509: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
[final_all_valid][11522 examples] EVAL Step loss is:  5.250361682706326 acc is: 0.4730081583058497 top-5 acc is: 0.7723485505988543
```