# Supplementary Materials for "MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks"

Ziyue Xiang[†] Amit Kumar Singh Yadav[†] Paolo Bestagini[‡] Stefano Tubaro[‡] Edward J. Delp[†]

[†]Video and Image Processing Lab (VIPER), School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, USA

[‡]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy

## 1. The Tree Restructuring Algorithm

The tree restructing algorithm is shown in Alg. 1. The `TreeRestructre` procedure is used on an MP4 tree $\mathcal{T}$ to generate the restructured tree $\mathcal{T}_r$.

## 2. String Processing

The string processing step consists of the following steps:

- Noise removal: non-printable and non-ASCII characters are removed from the input string
- String partition: the input string is broken down into parts separated by white space characters, where each part can be a textual word, URL, version string, ratio string, or numeric string
- URL decomposition: the input strings can also contain URLs which are decomposed into multiple parts; for example, `www.example.com/about/index.html` will be broken down into `www.example.com`, `about`, `index.html`
- Version string decomposition: version strings found in the input string are decomposed into multiple numeric parts; the numeric parts are surrounded by two special words `#VER_BEG` and `#VER_END` to emphasize that they are the members of a version string
- Ratio string conversion: ratio strings such as `1/48000` found in the input string are converted into their corresponding numeric values
- Numeric string conversion: string representations of floating point numbers and integers found in the input string are converted into their corresponding numeric values

## 3. MTN Model Details

### 3.1. Graph Pooling (GP) Readout

Denote the output of the Graph Pooling (GP) layer by $\boldsymbol{E}'$. The first dimension of $\boldsymbol{E}'$ is variable. To produce a fixed-sized vector for classification, a readout mechanism [1] is defined as follows:

$$\text{ReadOut}\left(\boldsymbol{E}'\right) = \text{MaxP}\left(\boldsymbol{E}'\right) \,||\, \text{MinP}\left(\boldsymbol{E}'\right). \tag{1}$$

Here, MaxP denotes the max pooling operation that determines the maximum element along each column of $\boldsymbol{E}'$, MinP denotes the min pooling operation, and $||$ denotes concatenation. The readout operation always produces a $2C$-dimensional vector, which can be used by a classifier. Here, $C$ is the number of "channels" in the Graph Convolution (GC) layer.

In MTN, we use the hierarchical pooling architecture described in [1]. That is, there are multiple GC+GP layers in the Graph Neural Network (GNN). The readout operation is used for each GP layer in the GNN, and the results are summed together to generate a single vector for classification.

### 3.2. Model Size of the Node Embedding Network (NEN)

The relationship between the choice of $M$ (embedding size) and the number of parameters in the Node Embedding Network (NEN) are shown in Tab. 1.

Table 1. The choice of $M$ and the corresponding size of the NEN.

| $M$ | 128 | 256 | 384 |
|---|---|---|---|
| Num. Parameters | 6.56M | 15.39M | 26.52M |

### 3.3. Model Architecture

We implemented MTN using the `PyTorch` library[1]. We used the GC and GP implementation from the `PyG` library[2].

We used `PyTorch`'s transformer encoder implementation (`TransformerEncoderLayer`). In the number encoder, we chose $\tau_{\text{max}} = 10^{12}$ and $\tau_{\text{min}} = -\tau_{\text{max}}$ (see Sec. 2.2.1). The String Object List (SOL) encoder in the NEN contains 4 transformer encoder layers with $M = 384$ and $h = 4$. The Graph Attention Network (GAtN) in the Graph Analysis Module (GAM) contains 6 transformer encoder layers with $M = 384$ and $h = 4$. The choice of $M$ is discussed in Sec. 4.1. For GC, we used the `GCNConv` layer from the `PyG` library with $C = 128$. For GP, we used the `SAGPool` layer from the `PyG` library with $\alpha = 0.5$. The definition of $C$ and $\alpha$ can be

[1] https://pytorch.org/docs/stable/index.html

[2] https://pytorch-geometric.readthedocs.io/en/latest/

**Algorithm 1:** The tree restructuring procedure.

**Procedure** `TreeRestructure(`$\mathcal{T}$`)`**:**
  **Input:** MP4 tree $\mathcal{T}$
  **Output:** restructured MP4 tree $\mathcal{T}_r$
  Create a new empty tree $\mathcal{T}_r$;
  `// Make sure the new tree` $\mathcal{T}_r$ `has the same topology as` $\mathcal{T}$
  **foreach** *node* $n$ *in* $\mathcal{T}$ **do** Create a new node $n'$ in $\mathcal{T}_r$ with tag `n` ;
  **foreach** *node* $n$ *in* $\mathcal{T}$ **do**
    **foreach** *child node* $c$ *of* $n$ **do**
      Find the corresponding nodes of $n$ and $c$ in $\mathcal{T}_r$, which are denoted by $n'$ and $c'$;
      Connect $n' \rightarrow c'$ in $\mathcal{T}_r$;
    **end**
  **end**
  `// Add data from` $\mathcal{T}$ `to` $\mathcal{T}_r$
  **foreach** *node* $n$ *in* $\mathcal{T}$ **do**
    Find the corresponding node of $n$ in $\mathcal{T}_r$, which is denoted by $n'$;
    **foreach** *key-value pair (key,val) in* $n$ **do** `RecursiveAdd(`*(key-val),* $n'$`)` ;
  **end**
  **return** $\mathcal{T}_r$
**Procedure** `RecursiveAdd(`*obj,* $u'$`)`**:**
  **Input:** an object to be added (*obj*), which can be a string, a number, a key-value pair, a list, a dictionary; the node where the data is added ($u'$)
  `// Recursively adds data into the restructured tree` $\mathcal{T}_r$
  **if** *obj is a number or a string* **then**
    Create a new node $v'$ in $\mathcal{T}_r$ with tag `v`;
    Set the data of $v'$ as *obj*;
    Connect $u' \rightarrow v'$ in $\mathcal{T}_r$;
  **else if** *obj is a key-value pair (key,val)* **then**
    Create a new node $k'$ in $\mathcal{T}_r$ with tag `k`;
    Set the data of $k'$ as *key*;
    Connect $u' \rightarrow k'$ in $\mathcal{T}_r$;
    `RecursiveAdd(`*val,* $k'$`)`
  **else if** *obj is a dictionary* **then**
    Create a new node $w'$ in $\mathcal{T}_r$ with tag `n`;
    Set the data of $w'$ as `#DICT`;
    Connect $u' \rightarrow w'$ in $\mathcal{T}_r$;
    **foreach** *(key,val) in obj* **do** `RecursiveAdd(`*(key,val),* $w'$`)`;
  **else if** *obj is a list* **then**
    Create a new node $w'$ in $\mathcal{T}_r$ with tag `l`;
    Set the data of $w'$ as `#LIST`;
    Connect $u' \rightarrow w'$ in $\mathcal{T}_r$;
    **foreach** *item in obj* **do** `RecursiveAdd(`*item,* $w'$`)`;

found in Sec. 2.3. The MTN has 3 GC+GP layeres. Under this setting, the GC+GP layers contain 281k parameters. For classification, we used a three layer multilayer perceptron network with $2C, 2C, K$ neurons. Here, $K$ is the number of classes in the video forensics task.

## References


[1] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention Graph Pooling. *Proceedings of 2019 International Conference on Machine Learning*, pages 3734–3743, 2019. Long Beach, CA, USA. 1