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

Abstract

MP4 video files are stored using a tree data structure. These trees contain rich information that can be used for forensic analysis. In this paper, we propose MP4 Tree Network (MTN), an approach based on an end-to-end Graph Neural Networks (GNNs) that is used for forensic analysis of MP4 trees. MTN does not use any video pixel data. MTN is trained using Self-Supervised Learning (SSL), which generates semantic-preserving node embeddings for the nodes in an MP4 tree. We also propose a data augmentation technique for MP4 trees, which helps train MTN in data-scarce scenarios. MTN achieves good performance across 3 video forensics tasks on the EVA-7K dataset. We show that MTN can gain more comprehensive understanding about the MP4 trees and is more robust to potential attacks compared to existing methods.

1. Introduction

The MP4 video container [13] is one of the most popular video container standards. MP4 files use a tree data structure to store information internally [13, 14]. In addition to video/audio bit streams which occupy most of the space in an MP4 file, other metadata information such as subtitles, codec type, video-audio synchronization information, timestamps, and geographic locations can also be stored. An MP4 file can be logically split into two components: 1) the "MP4 tree" that contains information related to both the tree topology and the metadata stored in the container; 2) the "bit streams" that contain information related to the encoded video pixel values and audio tracks. Existing work [1, 9, 32, 33] have shown that the MP4 tree (*i.e.* metadata information as well as the topology of the MP4 tree) can be used for video forensic analysis. This is a relatively new and developing concept since most existing Video Forensics Methods (VFMs) use the bit stream (*i.e.* pixel data) to make decisions in tasks such as camera model attribution, deepfake detection, and manipulation detection [2, 19, 29].

It has been shown in recent work that VFMs using the MP4 trees can provide an independent and alternative perspective compared to those that rely on pixel data (*i.e.* the bit streams). For example, Güera *et al.* [9] used metadata information extracted by the *ffprobe*[1] tool with Support Vector Machines (SVMs) and decision trees for video manipulation detection. Iuliani *et al.* [14] used hand-crafted features from the MP4 tree with a statistical classifier for video forensics analysis tasks. Yang *et al.* [33] converted the MP4 tree to an integer vector representation, which can be processed by decision tree classifiers for video forensics analysis tasks. Xiang *et al.* [32] extended the approach in [33] by improving MP4 tree parsing and introducing feature selection and dimensionality reduction to the vector representation of MP4 trees. Altinisik *et al.* [1] fused MP4 tree and video bit stream analysis for video source attribution.

In this paper, we propose MP4 Tree Network (MTN), which is a VFM based on end-to-end Graph Neural Networks (GNNs). MTN analyzes only the information in the MP4 tree (*i.e.* no bit stream data is used) and can be used to perform different forensic tasks.

In existing MP4 tree analysis [9, 32, 33], hand-crafted features with light-weight classifiers such as SVM [11] or decision trees [23] are used to predict the labels for the MP4 trees. Due to the limitations of hand-crafted features and light-weight classifiers, these methods are unable to process unseen data in MP4 trees and often fail to gain comprehensive understanding about the MP4 tree. MTN does not have these shortcomings as it is based on end-to-end deep neural networks. The scalability of deep neural networks also allows MTN to process very large datasets. We propose a Self-Supervised Learning (SSL) scheme for MTN, which enables it to generate semantic-preserving node embeddings for MP4 tree nodes. The design of the SSL scheme helps MTN cope with unseen data in MP4 trees. We also devise a data augmentation technique for MP4 trees, which helps train MTN in data-scarce scenarios.

We evaluate the performance of MTN on the EVA-7K dataset [33], where MTN shows good performance in 3 video forensics analysis tasks. We also show that MTN can gain a more comprehensive understanding on MP4 trees and is more

[1] https://ffmpeg.org/ffprobe.html

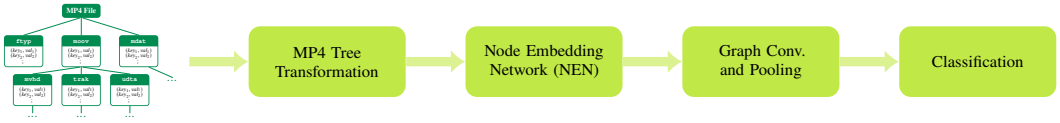

Figure 1. The block diagram of MP4 Tree Network (MTN).

robust to potential attacks compared to existing methods.

2. MP4 Tree Network (MTN)

The block diagram of MTN is shown in Fig. 1. The input MP4 tree is first transformed so that it can be processed by the Graph Neural Networks (GNNs) (Sec. 2.1). Then, the Node Embedding Network (NEN) is used to process the transformed MP4 tree to generate a node embedding for each node in the transformed tree (Sec. 2.2). After that, the Graph Convolution (GC) and Graph Pooling (GP) step analyzes all node embeddings and produce a graph embedding (Sec. 2.3), which is a single vector representation of the MP4 tree. Finally, the graph embedding vector is used by a classifier in the classification step to predict a label for the input MP4 tree depending on the specific forensic analysis task.

2.1. MP4 Tree Transformation

MP4 video files are organized using a tree data structure [13, 14] (as shown in Fig. 2). An MP4 video file is made up of a set of nodes, where each node contains the following information:

- The node type (*e.g.* `ftyp`, `moov`), which annotates the type of data stored in the node and its descendants.
- The node data, which carries the information in the node. Pieces of information are stored as a list of key-value pairs (*e.g.* (key_1, val_1), (key_2, val_2)). Each key is a string, while each value can be a string, number, list, or dictionary.
- The list of child nodes, which is used to maintain the tree structure.

Note that path-like strings can be used to point at nodes and node data in an MP4 tree. For example, `moov/udta` points to the `udta` node shown in Fig. 2. The string `moov/udta/@key2=val2` points to a key-value pair stored in the `udta` node. Here, `@` and `=` are the prefix and delimiter for key-value pairs, respectively.

Analyzing an MP4 tree is far from being a simple task, as the node data contains variable amount of key-value pairs, and the value of a pair can be another data structure such as list or dictionary. MP4 trees have to be transformed so that they can be processed by GNNs. The MP4 tree transformation process we propose takes as input an MP4 tree $\mathcal{T}$ and returns a transformed version of the tree $\mathcal{T}_t$. In $\mathcal{T}_t$, each node contains the following information:

- The tag, which indicates the type of node.
- The data, which is either a String Object List (SOL) or a number.
- The list of child nodes.

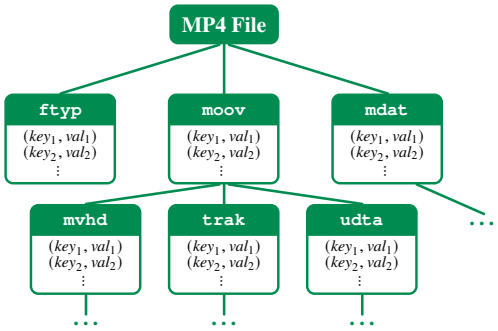

Figure 2. The illustration of the tree structure of MP4 video files.

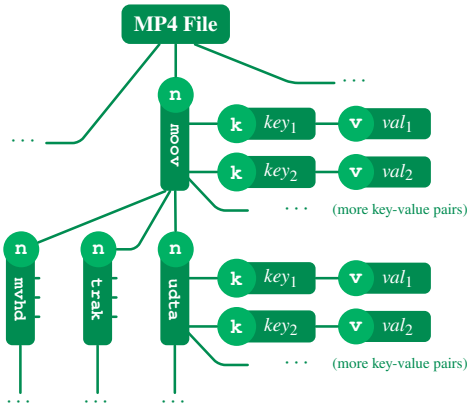

Figure 3. The restructured MP4 tree derived from the tree illustrated in Fig. 2. For each node, the tag is shown in the circle and the data is shown in the rectangle.

The MP4 tree transformation is a two-step process. In the first step, the MP4 tree $\mathcal{T}$ is restructured into a new tree $\mathcal{T}_r$. In the second step, the strings in $\mathcal{T}_r$ are processed to generate the transformed tree $\mathcal{T}_t$. We provide more details about these two steps.

2.1.1 MP4 Tree Restructuring

This step maps complicated data structures (*e.g.* list and dictionary) in an MP4 tree $\mathcal{T}$ to graph information that can be processed by GNNs. It makes sure that every node in the resulting tree $\mathcal{T}_r$ contains one string or one number as data.

In tree restructuring, we represent complicated data structures in the $\mathcal{T}$ using a series of nodes and connections in $\mathcal{T}_r$. For example, a key-value pair *(key,val)* in $\mathcal{T}$ can be represented by two interconnected nodes in $\mathcal{T}_r$, where the first node has tag `k` and data *key*, and the second node has tag

v and data *val*. A list in $\mathcal{T}$ can be represented by creating a new node ℓ' with tag l in $\mathcal{T}_r$ as the "header" of the list, and for each item in the list, create a child node of ℓ' with tag v that stores the item as its data . A precise definition of the tree restructuring procedure can be found in supplementary materials. In the restructured tree $\mathcal{T}_r$, each node is associated with one of the four tags k, v, l, or n. k nodes and v nodes are used to represent keys and values, respectively. A key-value pair in the original tree is represented as an edge between a k node and a v node. l nodes are list headers, whose child nodes are items in the list. n nodes represent nodes in the original MP4 tree as well as dictionary headers, whose child nodes are k nodes or n nodes. If all values in the MP4 tree shown in Fig. 2 are strings or numbers, then the corresponding restructured MP4 tree is shown in Fig. 3.

The restructured tree preserves almost all information in the original tree (only the order of list items is missing) while ensuring that the data of each node is either a string or a number. This makes it easier for GNNs to analyze the restructured MP4 trees.

2.1.2 String Processing

After the tree restructuring, the data of each node in $\mathcal{T}_r$ is either a string or a number. A string in $\mathcal{T}_r$ can be further divided to multiple parts that contain heterogeneous information. For example, the string Lavf58.29.100 indicates that the MP4 file contains the signature from the libavformat encoding library in ffmpeg[2] version 58.29.100. It is ideal for the analysis method to see this string represented as Lavf, *58*, *29*, *100* where Lavf is a string and *58*, *29*, *100* are numbers. The string processing step removes noise, partitions the string into "words", and distinguishes between textual data and numerical data. A detailed description of this step can be obtained from the supplementary materials.

This step converts each string in $\mathcal{T}_r$ into a list of objects known as SOLs, where each object in the SOL is either a textual word or a real number. The resulting tree is the transformed MP4 tree, which is denoted by $\mathcal{T}_t$. After this step, the data of each node in the transformed tree $\mathcal{T}_t$ is either an SOL or a number. SOLs are lists, but they do not make the structure of $\mathcal{T}_t$ more complicated. As described in Sec. 2.2, the information from all items in an SOL will be summarized into a single fixed-size vector.

2.2. Node Embedding Network (NEN)

Graph Neural Networks (GNNs) are neural networks that can analyze graph data structures [28, 31]. Since the transformed MP4 trees are a special case of generic graphs, we can use GNNs to analyze them.

The proposed Node Embedding Network (NEN) is based on GNNs. The NEN analyzes a transformed MP4 tree $\mathcal{T}_t$ and produce a node embedding for each node in $\mathcal{T}_t$. The block diagram of NEN is shown in Fig. 4. NEN is made up of three components: the number encoder, the SOL encoder, and the Graph Analysis Module (GAM).

The number encoder (Sec. 2.2.1) generates a vector representation for numbers in $\mathcal{T}_t$. The SOL encoder (Sec. 2.2.2) generates a vector for an SOL that summarizes information from all items in the SOL. Using the number encoder or SOL encoder, the data from the nodes in $\mathcal{T}_t$ are represented using real-valued vectors known as node data vectors. The node data vectors are processed by the GAM (Sec. 2.2.3) to produce the node embeddings, which are M-dimensional real-valued vectors. In Sec. 2.2.4, we describe how the NEN can be pretrained using Self-Supervised Learning (SSL) techniques so that the node embeddings contain semantic information about the MP4 tree.

Note that Natural Language Processing (NLP) approaches [21] are used in the NEN to process numbers and SOLs. The word embedding technique proposed in [5, 18] is used to convert words into word embeddings, which are M-dimensional vectors that can be updated during training. In the following, we use the arc symbol over a word ($\overbrace{\texttt{word}}$) to denote the embedding of the word.

2.2.1 Number Encoder

The role of the number encoder is to represent numbers as M-dimensional vectors. Although it is possible to use existing NLP approaches such as BERT [5] for this conversion, it has been shown in [26] that the number representations generated by most existing NLP approaches can be inefficient and can lead to worse performance in number-related reasoning tasks. Therefore, in the NEN, we explicitly represent numbers as the linear combination of two known vectors. This representation is more efficient since it only requires two M-dimensional vectors. The represented number can also be eailsy retrieved once the two vectors are given.

The number encoder operation is described as follows. It first converts the input number to symmetric log scale, which is defined as

$$\text{symlog}(x) = \text{sign}(x) \cdot \log_{10}\left(1 + |x \cdot \ln(10)|\right). \quad (1)$$

The value of symlog(x) preserves the sign and value of x while compressing the scale of x.

Define the clip operation as

$$\text{clip}(x, a, b) = \max\left[a, \min(x, b)\right], \quad (2)$$

where a is the lower bound and b is the upper bound. From the data, we determine the minimum and maximum numbers to be represented by the number encoder, which are denoted by $\tau_{\min}$ and $\tau_{\max}$, respectively. We add two special words #NUM1 and #NUM2 to the set of all words. Their corresponding word embeddings $\overbrace{\texttt{\#NUM1}}$ and $\overbrace{\texttt{\#NUM2}}$ are the vector representations of $\tau_{\min}$ and $\tau_{\max}$. All numbers between $\tau_{\min}$ and $\tau_{\max}$ can be represented as a linear combination of $\overbrace{\texttt{\#NUM1}}$ and $\overbrace{\texttt{\#NUM2}}$.

Some numbers (e.g., UNIX timestamps) in the MP4 tree are used to represent time stamps. To preserve the semantic meaning of such numbers and separate them from

[2] https://ffmpeg.org/

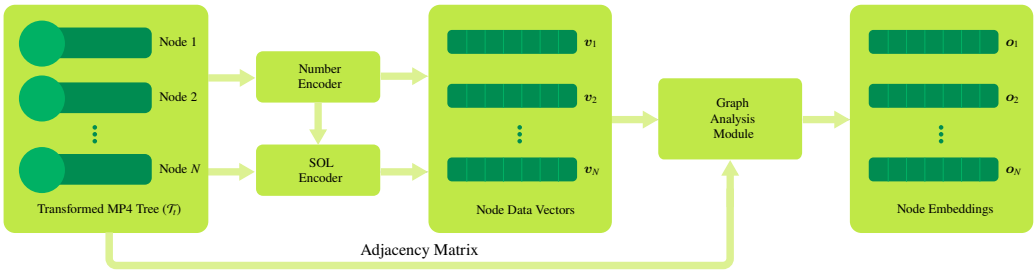

Figure 4. The block diagram of Node Embedding Network (NEN).

"plain" numbers, we introduce a time projection matrix $\boldsymbol{P}_{\text{time}} \in \mathbb{R}^{M\times M}$, which can be updated during training. As a result, the number encoder operation, denoted by $\text{Nenc}(\cdot)$, can be written as

$$\text{Nenc}(x) = \boldsymbol{P}\left(s(x)\,\overrightarrow{\texttt{\#NUM1}} + [1-s(x)]\,\overrightarrow{\texttt{\#NUM2}}\right), \tag{3}$$

where

$$s(x) = \frac{\text{symlog}\left[\text{clip}(x,\tau_{\min},\tau_{\max})\right] - \text{symlog}(\tau_{\min})}{\text{symlog}(\tau_{\max}) - \text{symlog}(\tau_{\min})}, \tag{4}$$

and

$$\boldsymbol{P} = \begin{cases} \text{identity} & \text{if } x \text{ is a plain number,} \\ \boldsymbol{P}_{\text{time}} & \text{if } x \text{ is a time stamp number.} \end{cases} \tag{5}$$

It can be seen that $\text{Nenc}(x) \in \mathbb{R}^M$.

2.2.2 SOL Encoder

As described in Sec. 2.1.2, a transformed MP4 tree contains SOLs. The role of the SOL encoder is to map each SOL into an M-dimensional vector.

Items in an SOL are either textual words or numbers. Denote an SOL of length L by $\{\rho_1,\ldots,\rho_L\}$. We first convert each item in the SOL into a vector representation using the following rule:

$$\boldsymbol{\rho}'_i = \begin{cases} \overrightarrow{\rho_i} + \boldsymbol{e}_i & \text{if } \rho_i \text{ is a textual word,} \\ \text{Nenc}(\rho_i) + \boldsymbol{e}_i & \text{if } \rho_i \text{ is a number,} \end{cases} \tag{6}$$

where $\boldsymbol{\rho}'_i \in \mathbb{R}^M$, and $\boldsymbol{e}_i \in \mathbb{R}^M$ is the positional embedding [6] for the i-th item in the SOL that can be updated during training. They added to the vector representations of the SOL items to encode the order of items. We also encode the length of the SOL by computing $\boldsymbol{\rho}'_0 = \text{Nenc}(L) + \boldsymbol{e}_0$.

We use the transformer encoder [27] to analyze the sequence of vectors $\{\boldsymbol{\rho}'_0, \boldsymbol{\rho}'_1, \ldots, \boldsymbol{\rho}'_L\}$. The transformer encoder, which is based on the Multihead Self Attention (MSA) mechanism, has been successful in many sequence processing tasks [5–7]. The output of the transformer encoder is a new sequence of vectors $\{\tilde{\boldsymbol{\rho}}'_0, \tilde{\boldsymbol{\rho}}'_1, \ldots, \tilde{\boldsymbol{\rho}}'_L\}$, $\tilde{\boldsymbol{\rho}}'_i \in \mathbb{R}^M$. The output of the SOL encoder is the average of the transformer encoder output. That is, the SOL encoder operation (denoted by $\text{Senc}(\cdot)$) can be written as

$$\text{Senc}\left(\{\rho_1,\ldots,\rho_L\}\right) = \frac{1}{L+1}\sum_{i=0}^{L}\tilde{\boldsymbol{\rho}}'_i. \tag{7}$$

After the SOL encoder operation, the information in the SOL is summarized in an M-dimensional vector.

2.2.3 Graph Analysis Module (GAM)

The Graph Analysis Module (GAM) analyzes the tag, data, and parent/child nodes of each node in the transformed MP4 tree. It consists of two components: node tag encoding and Graph Attention Network (GAtN). In this section, we assume there are N nodes in the transformed MP4 tree $\mathcal{T}_t$. The tag of each node is denoted by $t_1,\ldots,t_N$, where $t_i \in \{\texttt{k},\texttt{v},\texttt{l},\texttt{n}\}$. The data of each node is denoted by $d_1,\ldots,d_N$, where d_i is either an SOL or a number. The node data vector of the i-th node, denoted by $\boldsymbol{v}_i$, is given by

$$\boldsymbol{v}_i = \begin{cases} \text{Senc}(d_i) & \text{if } d_i \text{ is an SOL,} \\ \text{Nenc}(d_i) & \text{if } d_i \text{ is a number.} \end{cases} \tag{8}$$

Node tag encoding. The node data vectors $\boldsymbol{v}_i$ contain information about the node data. To encode the tag information of the nodes, we use the linear projection approach described in [31]. More concretely, we introduce four $M\times M$ matrices $\boldsymbol{P}_{\texttt{k}}$, $\boldsymbol{P}_{\texttt{v}}$, $\boldsymbol{P}_{\texttt{n}}$, and $\boldsymbol{P}_{\texttt{l}}$, which can be updated during training. We use them to generate the tag encoded node data vector $\boldsymbol{u}_i$ as follows:

$$\boldsymbol{u}_i = \boldsymbol{P}_{t_i}\boldsymbol{v}_i. \tag{9}$$

The vectors $\boldsymbol{u}_i$ are used as the input to the GAtN step.

Graph Attention Network (GAtN). GAtNs are based on the Graph Attention (GA) [28] technique, which is an adaptation to the MSA mechanism [27] so that graph-structured data can be processed. The MSA is an extension to Self Attention (SA). For both MSA and SA, the input is a sequence of vectors $\boldsymbol{w}_1,\ldots,\boldsymbol{w}_K$, and the output is a new sequence of vectors $\tilde{\boldsymbol{w}}_1,\ldots,\tilde{\boldsymbol{w}}_K$, where $\boldsymbol{w}_i, \tilde{\boldsymbol{w}}_i \in \mathbb{R}^M$. In SA, the similarity between all possible pairs $(\boldsymbol{w}_i, \boldsymbol{w}_j)$ in the input sequence is analyzed, and the output of the i-th element in the sequence $\tilde{\boldsymbol{w}}_i$ contains information about the relationship

between the i-th element and other elements in the sequence. In MSA, each element in the input sequence $\boldsymbol{w}_i$ is linearly projected into h lower dimensional sequences, where each element in the sequence has M/h dimensions. The SA is computed over the h lower dimensional versions of the input sequence, and the SA result from the h versions are merged together to produce the output sequence where each element has M dimensions.

GA incorporates the graph connectivity information in MSA so that the attention mechanism can be used to analyze graphs. The input to GA is a series of node data vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_K$, where $\boldsymbol{u}_i$ is the vector representation of the i-th node in the graph. The output of GA is a series of node embeddings $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_K$, and $\boldsymbol{u}_i, \boldsymbol{o}_i \in \mathbb{R}^M$. Unlike SA where the similarity between all possible pairs are analyzed, in GA, the similarity between $(\boldsymbol{u}_i, \boldsymbol{u}_j)$ is analyzed only when the i-th node and the j-th node are connected in the graph.

2.2.4 Self-Supervised NEN Pretraining

SSL refers to training machine learning models without data labels [12]. SSL techniques can obtain information and representation from unlabeled data, which can be helpful for a variety of downstream tasks. It has been used in NLP [5], image processing [20], and audio processing [7]. SSL can reduce the need of training data [7], increase model robustness [12], create semantically meaningful representations [20], and facilitate transfer learning [34]. We pretrain the NEN using SSL to generate node embeddings $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_N$, where $\boldsymbol{o}_i \in \mathbb{R}^M$ preserves semantic information about the i-th node. The pretrained NEN allows easier fine-tuning for various video forensics tasks.

To pretrain NEN, we devise three classification and two regression SSL "pretext" tasks. To enable SSL, we introduce a special word `#MASK`, whose word embedding $\overline{\texttt{\#MASK}}$ will be used to mask the node data vectors passed to the GAM. The five pretext tasks are described as below.

1. Masked node data type classification (DC): randomly select from the tree the i-th node and mask its data vector $\boldsymbol{u}_i$ with $\overline{\texttt{\#MASK}}$. Use the node embedding $\boldsymbol{o}_i$ to classify if the data of the i-th node is SOL or number.
2. Masked node tag classification (TC): randomly select from the tree the i-th node and mask its data vector $\boldsymbol{u}_i$ with $\overline{\texttt{\#MASK}}$. Use the node embedding $\boldsymbol{o}_i$ to classify the node tag (`k`, `v`, `l`, `n`).
3. Masked SOL word inclusion classification (IC): randomly select from the tree the i-th node whose data is SOL and mask its data vector $\boldsymbol{u}_i$ with $\overline{\texttt{\#MASK}}$. Then, select two words from the set of all words, where the first word is in the SOL of the i-th node and the second word is not. Classify if the two words are in the SOL of the i-th node using the node embedding $\boldsymbol{o}_i$.
4. Masked number regression (NR): randomly select from the tree the i-th node whose data is number and mask its data vector $\boldsymbol{u}_i$ with $\overline{\texttt{\#MASK}}$. Regress the symmetric log value of the number of the i-th node using the node embedding $\boldsymbol{o}_i$.
5. Masked SOL length regression (LR): randomly select from the tree the i-th node whose data is SOL and mask its data vector $\boldsymbol{u}_i$ with $\overline{\texttt{\#MASK}}$. Regress the length of the SOL using the node embedding $\boldsymbol{o}_i$.

The DC TC NR tasks examine if the NEN can gain understanding about the MP4 tree by asking it to fill in missing information in the node data vectors. The IC LR tasks examine if the node embeddings of SOL data preserve the information in the SOL.

The set of words and their word embeddings determined at training time may not include all words that can appear in an MP4 tree. To cope with unseen words, we introduce a special word `#UNKNOWN`. When a word in the MP4 tree has not been seen, we will use $\overline{\texttt{\#UNKNOWN}}$ as its word embedding. To emulate the presence of unseen words during training, in the NEN pretraining, we randomly select 5% of the words in the transformed tree and replace their word embeddings with $\overline{\texttt{\#UNKNOWN}}$.

2.3. Graph Convolution and Graph Pooling

In popular Convolution Neural Network (CNN) based image classification techniques such as VGG [24] and ResNet [10], convolution layers are used to extract features from the input image and pooling layers are used to downsample the extracted features so that subsequent layers can focus on higher level features. Similar approach can be used for graphs using Graph Convolution (GC) and Graph Pooling (GP) techniques. The node embeddings generated by the NEN trained with SSL contains semantic information about the nodes in the MP4 tree. Processing the node embeddings with GC and GP techniques allows the neural network to focus on node data and graph topology that have significant contribution to the decision. In MTN, we follow the GC technique proposed by Kipf *et al.* [15] and the GP technique proposed by Lee *et al.* [16].

In GC, the node embedding is first stored in a matrix $\boldsymbol{E} \in \mathbb{R}^{N \times M}$, where N is the number of nodes and M is the dimensionality. The GC operation is defined as

$$\mathrm{GC}(\boldsymbol{E}) = \mathrm{ReLU}\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{E}\boldsymbol{\theta}\right), \tag{10}$$

where $\boldsymbol{D} \in \mathbb{R}^{N \times N}$ is the degree matrix, $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $\boldsymbol{\theta} \in \mathbb{R}^{M \times C}$ is the parameters of GC which can be updated during training. Here, C is a hyperparameter that determines the number of "channels" of GC. It can be seen that $\mathrm{GC}(\boldsymbol{E}) \in \mathbb{R}^{N \times C}$.

In GP, the graph is "downsampled" by a factor of $\alpha \in (0, 1)$. At first, a graph SA score $\boldsymbol{Z} \in \mathbb{R}^N$ is computed by

$$\boldsymbol{Z} = \tanh\left(\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}\,\mathrm{GC}(\boldsymbol{E})\boldsymbol{\theta}_{\mathrm{att}}\right), \tag{11}$$

where $\boldsymbol{\theta}_{\mathrm{att}} \in \mathbb{R}^C$ is the parameters of GP which can be updated during training. Then, the nodes associated with the top $\lceil \alpha N \rceil$ scores in $\boldsymbol{Z} \in \mathbb{R}^N$ are preserved in the graph, while

the remaining nodes are removed from the graph. That is, the output of GP (denoted by $\boldsymbol{E}'$) is a $\lceil \alpha N \rceil \times C$ matrix, whose rows are from those in GC($\boldsymbol{E}$) associated with top scores. After node removal, the degree matrix $\boldsymbol{D}$ and adjacency matrix $\boldsymbol{A}$ need to be updated accordingly.

3. Data Augmentation

Video data files are usually significantly larger compared to image, audio, and text data. Therefore, it is difficult to create, transmit, and store large video datasets. In practice, we need to analyze video datasets that are several gigabytes in size, but only contain several hundred samples per label. The lack of data can be challenging for the training of deep neural network models. To cope with this problem, we devise a data augmentation technique for training MTN.

The proposed data augmentation technique is derived from decision tree analysis for MP4 trees [33]. In this class of methods, the nodes and key-value pairs in an MP4 tree are first converted into path-like strings (as described in Sec. 2.1). Then, the occurrence of each unique string is counted throughout the tree, which effectively creates an integer-valued vector representation (denoted by $\boldsymbol{q}$) for the MP4 tree. The vector $\boldsymbol{q}$ is used by decision tree classifiers to predict labels for the MP4 tree. To classify an MP4 tree based on $\boldsymbol{q}$, the decision tree must have split on a number of elements in $\boldsymbol{q}$, where each split either indicates the presence of a string or the absence of a string. We focus on the splits that correspond to the presence of a string and denote such strings by $s_1, \ldots, s_R$. Each path-like string s_i represents a path in the MP4 tree from the root node to an inner node or a leaf node that contributed to the classification of the MP4 tree. Therefore, we mark all nodes in the MP4 tree along the path as *visited*. When the node marking procedure is done for strings $s_1, \ldots, s_R$, the nodes in the MP4 tree not marked as visited are less likely to have contribution to the decision process, which implies that they can be removed from the tree.

The splits from one decision tree may overfit the dataset and be subject to the noise in the data. To mitigate this issue, the proposed data augmentation technique is based on a random forest [3] trained using $\boldsymbol{q}$. Random forests are ensemble classifiers that aggregates the decisions from many decision trees (in our experiments, we used 100 decision trees). For each decision tree in the random forest, if it classifies the label correctly, we collect the strings s_i as described above and add the strings to the set of all strings. The node marking procedure is used for the set of all strings. For the nodes that are not marked as visited, we randomly remove β percent of them as the data augmentation step. In our experiments, we set $\beta = 30$. The data augmentation is not enabled when the random forest cannot classify the MP4 tree correctly, as the evidence used by the random forest can be unreliable.

4. Experiments and Results

In this section, we describe the data, experiments, and results. Throughout the experiments, we extracted the MP4 tree information from MP4 files using the technique described in [32]. We used F_1-score [22] and balanced accuracy [4] to measure the performance of various approaches.

4.1. NEN Pretext Tasks

The NEN was pretrained on the MFC dataset [8], which contains 4038 MP4 video files. In the NEN, the SOL encoder consists of 4 transformer encoder layers; the GAtN consists of 6 GA layers. We set the number of heads in the MSA (*i.e.* h) to be 4. Details about the NEN architecture can be obtained from the supplementary materials.

We pretrained the NEN using the AdamW optimizer [17] with an initial learning rate of 10^{-5} and a weight decay factor of 0.001. The batch size was set to 16. The classification pretext tasks were trained using the cross entropy loss [30]; the regression pretext tasks were trained using the mean squared error loss [30]. For each batch, the five tasks were trained sequentially. That is, we computed the loss with respect to the first task and updated the model before proceeding to the second task. The training was stopped after 30 epochs.

For the NEN, we need to determine two hyperparameters M (the dimensionality of embeddings) and γ (number of masked nodes in each graph). To determine the choice of parameters, we sampled combinations of (M, γ) and pretrained the NEN. We evaluated the training performance of each task on the MFC dataset after each epoch. The classification tasks were evaluated using accuracy [25] and regression tasks were evaluated using mean squared error [30]. We first fixed $\gamma = 24$ and chose M among 128, 256, and 384. The training performance of NEN is shown in Fig. 5. It can be seen that using $M = 384$ resulted in the best performance in (IC). Then, we fixed $M = 384$ and chose γ between 16 and 32. The performances are also shown in Fig. 5. It can be seen that using $\gamma = 16$ resulted in the best performance in (NR). Therefore, in further experiments, we selected $M = 384$ and $\gamma = 16$.

From Fig. 5, it can also be seen that the performance of all five tasks were generally improving as the number of epochs increased. This shows that the design of the five tasks allows them to function collaboratively to help the NEN learn from unlabeled MP4 trees.

4.2. Forensics Analysis Using MTN

For video forensics analysis tasks, the experiments were conducted on the EVA-7K dataset [33]. The dataset consists of 7000 MP4 video sequences captured by different mobile devices. There are 140 pristine video sequences in the dataset, which are the original video files captured by the mobile devices. All 140 pristine video sequences are manipulated using off-the-shelf tools such as *ffmpeg*[3], *Kdenlive*[4], and

[3] https://ffmpeg.org/

[4] https://kdenlive.org/en/

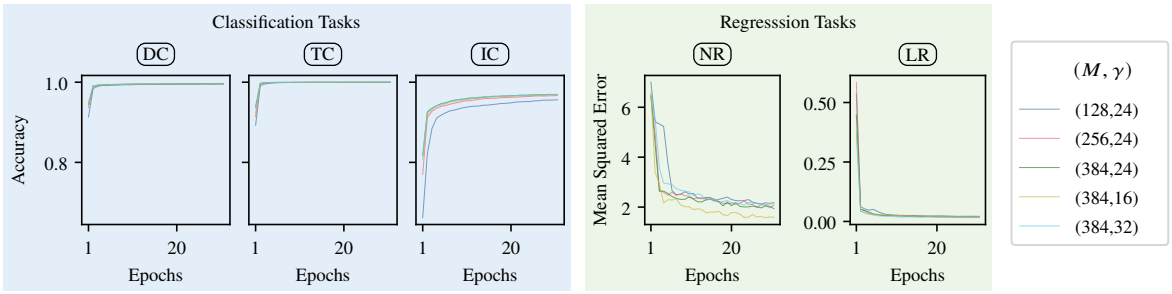

Figure 5. The training performance of pretext tasks using different (M, γ) settings.

Adobe Premiere[5]. This results in 1260 manipulated videos. The pristine and manipulate video sequences are uploaded to four social networks (YouTube, Facebook, TikTok, and Weibo) and then downloaded back. In the end, this results in 7000 video sequences.

We evaluated the performance of MTN in 3 video forensics analysis tasks: social network attribution, editing tool attribution, and manipulation detection. In each task, the EVA-7K dataset was divided into training, validation, and testing sets, with a ratio of 4:2:4.

We used the pretrained NEN with $M = 384$ and $\gamma = 16$ (see Sec. 4.1). We used three GC+GP layers with $C = 128$ and $\alpha = 0.5$ (see Sec. 2.3). The classification step of MTN is realized with a three-layer multilayer perceptron network. Details about the MTN architecture can be obtained from the supplementary materials.

For each training batch, we first sampled 16 MP4 trees from the training set. These 16 samples were added to the training batch. Then, for each sample, if it was classified correctly by the random forest approach (see Sec. 3), we applied data augmentation to the sample twice and put the two output MP4 trees in the training batch. Therefore, the batch size varies between 16 and 48. The MTN was trained using the AdamW optimizer [17] with a learning rate of 3×10^{-5} and a weight decay factor of 0.001. The training was stopped when the validation performance no longer increased. We report the performance of MTN on the testing set. The performance of MTN in each forensics task is reported as below.

- Social network attribution: in this task, the Video Forensics Method (VFM) is given videos downloaded from four social networks as well as local videos that are not from social networks. The VFM is asked to predict the source of the video, which can be either one of the four social networks or local. The performance comparison of this task is shown in Tab. 1.
- Editing tool attribution: in this task, the VFM is given videos that are edited by five video editing tools as well as unedited videos. The VFM is asked to predict which tool is used to edit the video, which can be either one of the five tools or unedited. The performance comparison of this task is shown in Tab. 2.
- Manipulation detection: in this task, the VFM is given videos that are manipulated by video editing tools as well as pristine videos. The VFM is asked to predict whether the video is manipulated. The performance comparison of this task is shown in Tab. 3.

[5] https://www.adobe.com/products/premiere.html

Table 1. The F_1-score comparison of the social network attribution task. The performance of "(Local)" category is not available for Yang *et al.* [33].

Social Network	Yang *et al.* [33]	Xiang *et al.* [32]	MTN
YouTube	1.00	0.99	1.00
Facebook	1.00	1.00	1.00
WeiBo	0.99	0.99	0.99
TikTok	1.00	1.00	1.00
(Local)	–	0.99	0.99

Table 2. The F_1-score comparison of the editing tool attribution task.

Tool	Yang *et al.* [33]	Xiang *et al.* [32]	MTN
Avidemux	0.99	0.98	0.99
Exiftool	0.98	1.00	0.96
ffmpeg	0.94	1.00	1.00
Kdenlive	0.95	1.00	0.99
Premiere	1.00	0.99	1.00
(Unedited)	0.97	1.00	0.96

Table 3. The balanced accuracy score comparison of the manipulation detection task.

	Balanced Accuracy
Güera *et al.* [9]	0.67
Iuliani *et al.* [14]	0.85
Yang *et al.* [33]	0.98
Xiang *et al.* [32]	0.99
MTN	1.00

It can be seen that MTN achieved the best performance in social network attribution and manipulation detection. The performance of MTN is in line with that of [32]. However, as we will discuss in Sec. 4.2.1, MTN has more comprehensive understanding about MP4 trees and is more robust against attacks.

4.2.1 The Robustness of MTN

Existing MP4 tree analysis approaches such as [32, 33] use hand-crafted features with decision tree classifiers. Decision

Table 4. The robustness of random forest and MTN against the evidence removal attack.

Method	Avg. Confidence Before Attack	Confidence Score Change: Num. Evidence Removal 1	2	3	4	5
Random Forest	0.99	-0.08	-0.17	-0.24	-0.31	-0.38
MTN	0.97	-0.08	-0.14	-0.18	-0.20	-0.21

tree classifiers function by analyzing a series of deterministic splits on selected elements in the hand-crafted feature vector. Therefore, this class of methods cannot model the topology of the MP4 tree or node-to-node relationships in the tree, which makes them more vulnerable against attacks. In this section, we demonstrate an evidence removal attack dedicated to methods for MP4 tree analysis based on decision trees, and we show that MTN is more robust against this attack.

In evidence removal attack, the attacker attempts to remove information from the MP4 tree in order to change the predicted label from VFMs. For the attack, we trained a random forest classifier for the social network attribution task that consists of 100 decision trees. For a given sample that is classified correctly by the random forest, we focus on the decision trees in the random forest that also classifies the sample correctly. We gathered all the splits in the focused decision trees that correspond to the presence of a node or key-value pair in the MP4 tree. This effectively generates a set of node and key-value pairs that serves as evidence for the decision. The set of evidence usually contains more than 20 elements. From the set of evidence, we randomly remove $1, 2 \ldots, 5$ node(s)/key-value pair(s) from the MP4 tree and ask both the random forest and MTN to classify the MP4 tree with evidence removal.

Both the random forest and MTN can output a confidence score for the prediction, which is the probability that the sample belongs to the ground truth label. By comparing the confidence score of the MP4 tree before and after evidence removal, we can evaluate the robustness of the classifiers against the evidence removal attack. The evidence removal attack was applied to all MP4 trees in the testing set, and we computed the average confidence score change caused by the attack.

The result is shown in Tab. 4. It can be seen that the confidence score decline of MTN is less significant compared to the random forest. Note that as the number of evidence removal increases, the confidence score from random forest continues to drop, whereas the confidence score from MTN almost stopped decreasing. This implies that the decision of MTN is likely to be based on a more comprehensive analysis of the MP4 tree, which makes MTN more robust against evidence removal attacks.

4.3. Ablation Study

We analyzed the effectiveness of NEN pretrain and data augmentation. We trained MTN on the social network attribution task with four settings: with both techniques (+Both); with only data augmentation (-NEN pretrain); with only NEN pretrain (-Data Aug.); with none of the techniques (-Both). We monitored the validation balanced accuracy of the four settings every 3 training steps. The result is shown in Fig. 6. It can be seen that NEN pretrain and data augmentation combined led to faster training as well as less performance fluctuation. As more training steps elapsed, the MTN trained using both techniques also managed to achieve the best validation balanced accuracy, which was close to 1.

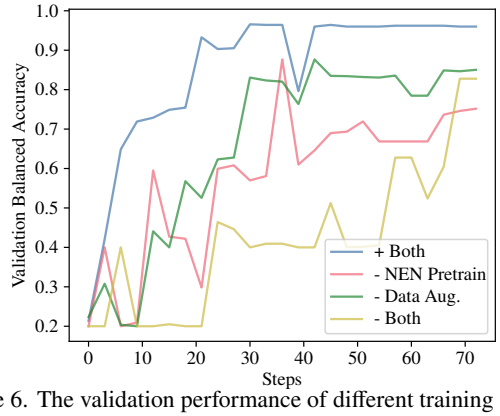

Figure 6. The validation performance of different training settings vs. training steps.

5. Conclusion

In this paper, we propose MTN for video forensics analysis based on end-to-end GNNs. MTN uses the MP4 tree information to make decisions, it does not require any pixel data. We devise an SSL scheme to pretrain MTN, which allows it to generate semantic-preserving node embeddings. We also propose a data augmentation technique for MP4 trees to help train MTN in data-scarce scenarios. The experimental results showed that MTN achieved good performance in video forensics analysis tasks. It is shown that MTN can gain more comprehensive understanding about the MP4 trees and is more robust to the evidence removal attack compared to existing methods. We also demonstrated that the SSL scheme and the data augmentation technique can reduce training iterations, reduce performance fluctuation, and improve the performance. The source code of MTN is avaliable at https://gitlab.com/viper-purdue/mtn.

In the future, we will examine the performance of MTN in more video forensics analysis tasks. We will evaluate the scalability of MTN by analyzing large real-world datasets.

Acknowledgements This material is based on research sponsored by DARPA and Air Force Research Laboratory (AFRL) under agreement number FA8750-20-2-1004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA and Air Force Research Laboratory (AFRL) or the U.S. Government. Address all correspondence to Edward J. Delp, ace@ecn.purdue.edu.

References

[1] Enes Altinisik, Hüsrev Taha Sencar, and Diram Tabaa. Video Source Characterization Using Encoding and Encapsulation Characteristics. *IEEE Transactions on Information Forensics and Security*, 17:3211–3224, 2022. 1

[2] Kratika Bhagtani, Amit Kumar Singh Yadav, Emily R Bartusiak, Ziyue Xiang, Ruiting Shao, Sriram Baireddy, and Edward J Delp. An Overview of Recent Work in Media Forensics: Methods and Threats. *arXiv preprint arXiv:2204.12067*, 2022. 1

[3] Gérard Biau and Erwan Scornet. A Random Forest Guided Tour. *Test*, 25:197–227, 2016. 6

[4] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. *Proceedings of the 20th International Conference on Pattern Recognition*, pages 3121–3124, 2010. Istanbul, Turkey. 6

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018. 3, 4, 5

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929*, 2020. 4

[7] Yuan Gong, Cheng-I Lai, Yu-An Chung, and James Glass. SSAST: Self-supervised Audio Spectrogram Transformer. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(10):10699–10709, 2022. Virtual. 4, 5

[8] Haiying Guan, Mark Kozak, Eric Robertson, Yooyoung Lee, Amy N Yates, Andrew Delgado, Daniel Zhou, Timothee Kheyrkhah, Jeff Smith, and Jonathan Fiscus. MFC datasets: Large-scale Benchmark Datasets for Media Forensic Challenge Evaluation. *Proceedings of the 2019 IEEE Winter Applications of Computer Vision Workshops*, pages 63–72, 2019. Waikoloa, HI, USA. 6

[9] David Güera, Sriram Baireddy, Paolo Bestagini, Stefano Tubaro, and Edward J Delp. We Need No Pixels: Video Manipulation Detection Using Stream Descriptors. *arXiv preprint arXiv:1906.08743*, 2019. 1, 7

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Proceedings of the 2016 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. Las Vegas, NV, 2016. 5

[11] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support Vector Machines. *IEEE Intelligent Systems and Their Applications*, 13(4):18–28, 1998. 1

[12] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty. *Advances in Neural Information Processing Systems*, 32, 2019. 5

[13] International Organization for Standardization. ISO/IEC 14496–14:2020–Information Technology–Coding of Audio-visual Objects–Part 14: MP4 File Format. https://www.iso.org/standard/79110.html, 2020. 1, 2

[14] Massimo Iuliani, Dasara Shullani, Marco Fontani, Saverio Meucci, and Alessandro Piva. A Video Forensic Framework for the Unsupervised Analysis of MP4-Like File Container. *IEEE Transactions on Information Forensics and Security*, 14(3):635–645, 2019. 1, 2, 7

[15] Thomas N Kipf and Max Welling. Semi-supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907*, 2016. 5

[16] 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. 5

[17] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6, 7

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013. 3

[19] Simone Milani, Marco Fontani, Paolo Bestagini, Mauro Barni, Alessandro Piva, Marco Tagliasacchi, and Stefano Tubaro. An Overview on Video Forensics. *APSIPA Transactions on Signal and Information Processing*, 1:e2, 2012. 1

[20] Ishan Misra and Laurens van der Maaten. Self-supervised Learning of Pretext-invariant Representations. *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717, 2020. Virtual. 5

[21] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2020. 3

[22] David M. W. Powers. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011. 6

[23] J. Ross Quinlan. Induction of Decision Trees. *Machine learning*, 1:81–106, 1986. 1

[24] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[25] Alaa Tharwat. Classification Assessment Methods. *Applied Computing and Informatics*, 17(1):168–192, 2021. 6

[26] Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski. Representing Numbers in NLP: a Survey and a Vision. *arXiv preprint arXiv:2103.13136*, 2021. 3

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. *Advances in Neural Information Processing Systems*, 30, 2017. 4

[28] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph Attention Networks. *arXiv preprint arXiv:1710.10903*, 2017. 3, 4

[29] Luisa Verdoliva. Media Forensics and DeepFakes: An Overview. *IEEE Journal of Selected Topics in Signal Processing*, 14(5):910–932, 2020. 1

[30] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science*, 9(2):187–212, 2022. 6

[31] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous Graph Attention Network.

Proceedings of the 2019 World Wide Web Conference, pages 2022–2032, 2019. San Francisco, CA, USA. 3, 4

[32] Ziyue Xiang, János Horváth, Sriram Baireddy, Paolo Bestagini, Stefano Tubaro, and Edward J Delp. Forensic Analysis of Video Files Using Metadata. *Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1042–1051, 2021. Virtual. 1, 6, 7

[33] Pengpeng Yang, Daniele Baracchi, Massimo Iuliani, Dasara Shullani, Rongrong Ni, Yao Zhao, and Alessandro Piva. Efficient Video Integrity Analysis Through Container Characterization. *IEEE Journal of Selected Topics in Signal Processing*, 14(5):947–954, 2020. 1, 6, 7

[34] Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain Autoencoders: Unsupervised Learning by Cross-channel Prediction. *Proceedings of the 2017 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1058–1067, 2017. Honolulu, HI, USA. 5