FastForensics: Efficient Two-Stream Design for Real-Time Image Manipulation Detection

Yangxiang Zhang[1], Yuezun Li[1,*], Ao Luo[2], Jiaran Zhou[1], Junyu Dong[1]

[1]School of Computer Science and Technology, Ocean University of China [2]Southwest Jiaotong University

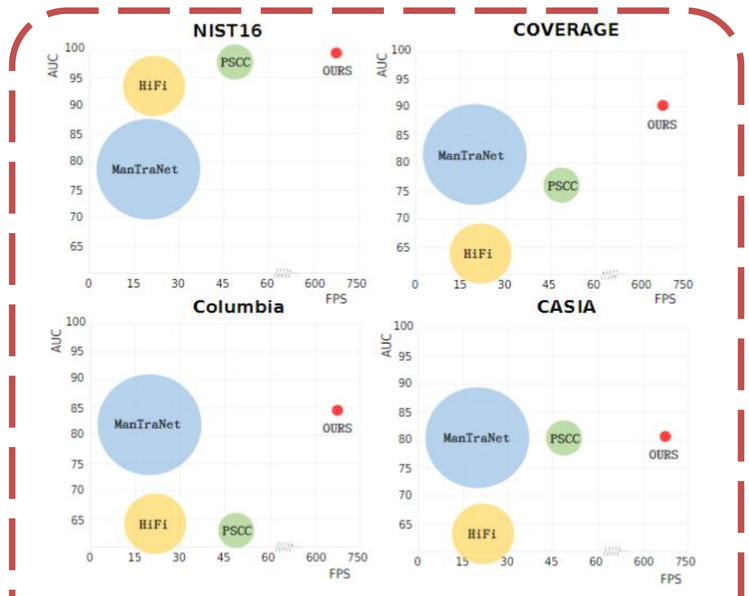
* Corresponding author

Introduction:

- In this paper, we describe an efficient two-stream architecture for real-time image manipulation detection.
- We propose a new Efficient Wavelet-guided Transformer Block (EWTB) that can integrate the wavelet transformation with an efficient and interactive design to highlight the manipulation traces.

Motivation:

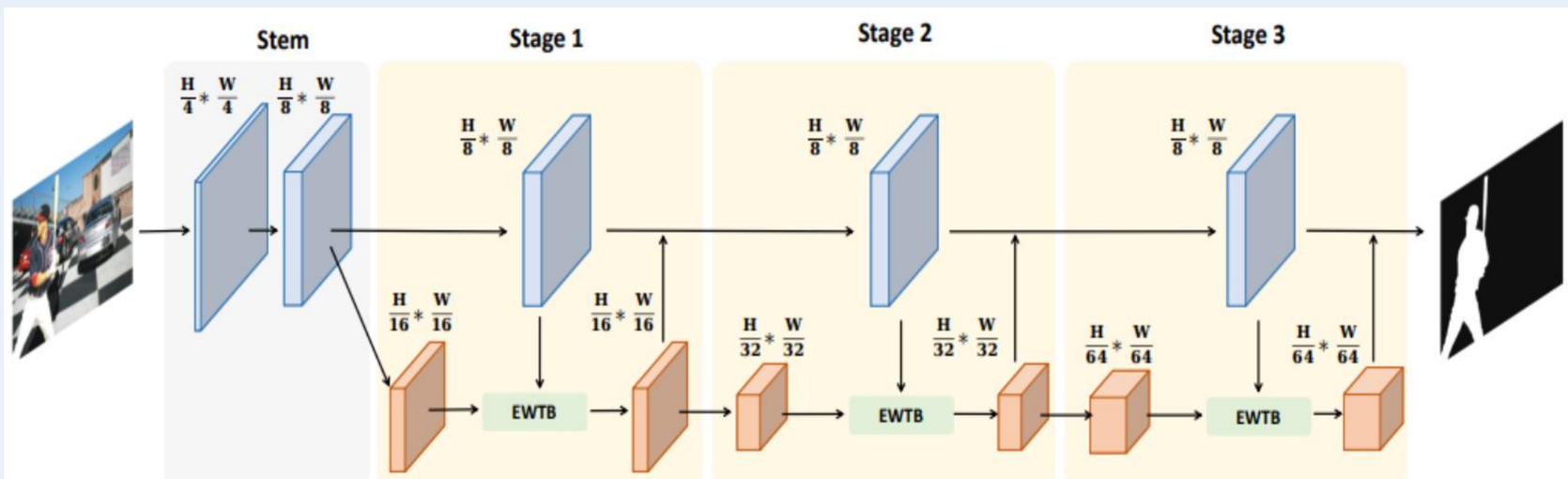
Previous methods have shown promising detection performance on many datasets, aligning with the current focus of the field on pursuing detection accuracy. However, as image manipulation techniques become more popular and social apps become more prevalent, accuracy alone is no longer the only criterion for detection methods.



AUC (%), FLOPs (circle area), FPS of different methods. Our method (red circle) strikes a good balance between performance and efficiency.

Two-Stream Design:

The cognitive branch aims to comprehend the context at a glance and determine where the manipulated regions exist.
The inspective branch is composed of simple convolution layers that focus on local detail and extract the specific shape of the manipulated regions.



Overview of the architecture of our method. The blue and orange cuboids represent the features from the inspective branch and the cognitive branch. EWTB is denoted as an Efficient Waveletguided Transformer Block.

Efficient Wavelet-guided Transformer Block:

- **Why is our self-attention fast?**

In vanilla self-attention, the major computational cost derives from the heavy linear transformations for query, key, and value features, we decompose the input features into different pieces and perform self-attention only on the respective piece.

- **Why use our Shared Global Queries?**

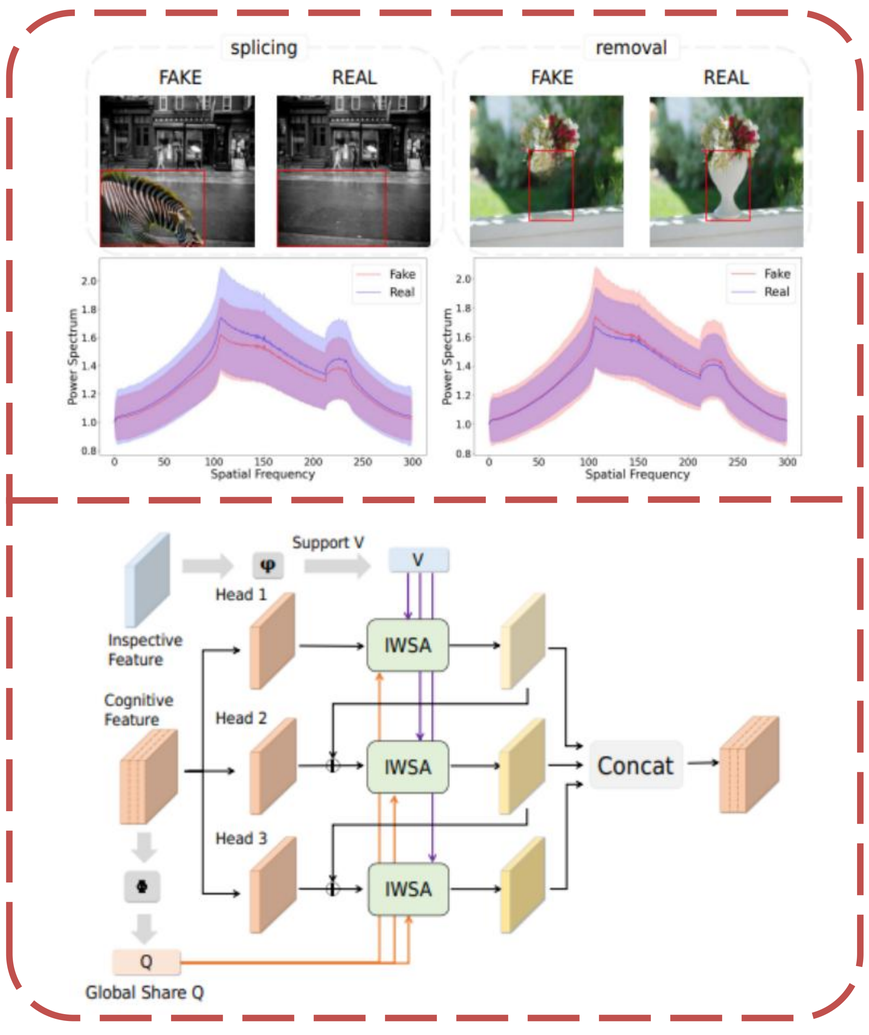
Considering that queries are important features that can provide sufficient indexing information, we formulate a shared global query for all heads, to better utilize the global information while further reducing the computation costs.

- **Why use our Support Value?**

The knowledge from the inspective branch can provide fine-grained information support for our proposed self-attention module.

- **Why use frequency domain information in self-attention?**

see figure on the right(top).



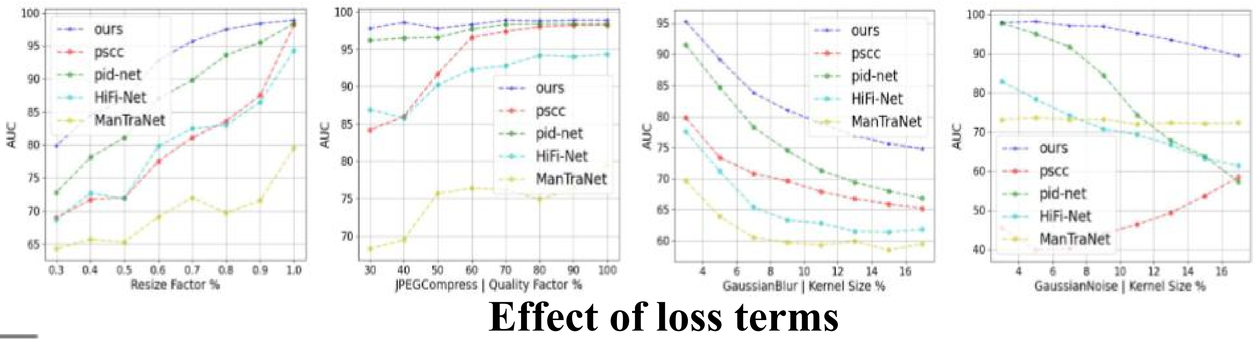
Experiments:

Datasets: CASIA, NIST16, Columbia, and Coverage

Results on four datasets:

Method	FLOPs ↓	Params ↓	FPS ↑	CASIA		Columbia		NIST16		COVERAGE		**Average**	
				F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑
ManTraNet* [27]	191.07G	3.80M	16	—	81.7	—	82.4	—	79.5	—	81.9	—	81.4
ManTraNet† [27]	191.07G	3.80M	16	10.5	77.2	19.1	73.3	7.1	76.3	5.4	73.7	10.5	75.1
PSCC [15]	20.38G	2.75M	48	44.5	80.5	16.4	52.9	68.7	98.2	33.7	76.7	40.8	77.1
HiFi-Net [7]	82.3G	10.2M	20	17.9	63.7	42.1	62.1	24.6	94.3	23.6	60.2	-	-
WSCL* [36]	>4.12G	>25.6M	<240	15.3	-	36.2	-	9.9	-	20.1	-	20.4	-
SAFL-Net* [21]	>5.35G	>66M	<80	74.0	90.8	-	96.9	87.9	99.7	80.3	97.0	80.7	96.1
ObjectFormer* [25]	>3.16G	>38M	<120	57.9	88.2	-	95.5	82.4	99.6	75.8	95.7	72.0	94.7
UEN* [12]	>34.6G	>45.2M	<48	62.9	87.7	-	-	93.2	99.6	-	-	78.0	93.6
BeSiNetV1 [32]	2.84G	13.27M	728	33.1	77.9	37.4	89.5	83.1	98.4	37.2	83.3	47.7	87.3
BeSiNetV2 [33]	2.36G	3.34M	600	31.9	75.9	28.4	81.8	82.1	98.0	32.9	77.3	43.8	83.3
PIDNET [29]	4.29G	28.53M	432	35.4	75.4	9.1	67.6	87.9	98.4	36.3	77.8	42.2	79.8
SegNeXt [6]	2.93G	13.89M	248	9.4	68.0	33.1	82.7	62.9	94.9	25.2	74.7	32.7	80.1
EfficientViT [1]	1.73G	15.27M	384	35.6	76.0	34.5	83.0	87.0	98.8	39.7	85.5	49.2	85.8
Segformer [28]	3.43G	24.72M	208	7.5	72.6	52.7	90.9	59.6	96.4	18.1	81.2	34.5	85.3
DDRNet [9]	3.57G	20.30M	536	18.6	74.6	41.6	85.9	28.8	82.3	32.5	82.1	30.4	81.2
Ours	2.16G	8.36M	672	44.6	81.9	49.3	84.7	86.5	98.9	54.7	90.6	58.8	89.0

Robustness analysis of different methods:



Effect of components

WL	SQ	SV	CASIA		NIST16		COVERAGE		Average	
			F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑
-	-	-	36.4	78.1	82.8	98.7	39.0	83.2	52.7	86.6
✓	-	-	36.2	80.4	85.4	98.6	38.8	83.0	53.5	87.3
✓	✓	-	37.6	78.1	81.4	98.7	43.4	86.0	54.1	87.6
✓	✓	✓	**44.6**	**81.9**	**86.5**	98.9	**54.7**	**90.6**	**61.9**	**90.4**

Effect of loss terms

L_{ce}	L_{bry}	L_{pos}	CASIA		NIST16		COVERAGE		Average	
			F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑	F_1 ↑	AUC↑
✓	-	-	37.4	79.7	86.4	98.9	38.8	84.1	54.2	87.6
✓	✓	-	41.1	80.7	85.5	98.7	47.8	87.6	58.1	89.0
✓	-	✓	39.6	81.2	81.3	98.9	42.6	82.8	54.5	87.6
✓	✓	✓	**44.6**	**81.9**	**86.5**	**98.9**	**54.7**	**90.6**	**61.9**	**90.4**