VIDEO CODEC FORENSICS BASED ON CONVOLUTIONAL NEURAL NETWORKS

S. Verde[1]*, L. Bondi*[2]*, P. Bestagini*[2]*, S. Milani*[1]*, G. Calvagno*[1]*, S. Tubaro*[2]

[1]Department of Information Engineering, University of Padova, Padova, Italy
[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy

ABSTRACT

The recent development of multimedia has made video editing accessible to everyone. Unfortunately, forensic analysis tools capable of detecting traces left by video processing operations in a blind fashion are still at their beginnings. One of the reasons is that videos are customary stored and distributed in a compressed format, and codec-related traces tends to mask previous processing operations.

In this paper, we propose to capture video codec traces through convolutional neural networks (CNNs) and exploit them as an asset. Specifically, we train two CNNs to extract information about the used video codec and coding quality, respectively. Building upon these CNNs, we propose a system to detect and localize temporal splicing for video sequences generated from the concatenation of different video segments, which are characterized by inconsistent coding schemes and / or parameters (e.g., video compilations from different sources or broadcasting channels). The proposed solution is validated using videos at different resolutions (i.e., CIF, 4CIF, PAL and 720p) encoded with four common codecs (i.e., MPEG2, MPEG4, H264 and H265) at different qualities (i.e., different constant and variable bitrates, as well as constant quantization parameters).

***Index Terms*—** Video forensics, video codec identification, temporal splicing, forgery detection, deep learning.

1. INTRODUCTION

Editing a video sequence in a realistic fashion is progressively becoming an easy task. On one hand, this is due to the large amount of powerful yet user-friendly editing softwares that are available on the market (e.g., Adobe Premiere, Apple Final Cut, etc.). On the other hand, this is also due to the huge recent advancements in computer vision and deep learning fields, that allow the creation of new impressive solutions for automatic video editing (e.g., FaceSwap, DeepFakes, etc.). Although these tools have a positive impact on video editing, film making, and artistic fields, the ability of easily forging a video sequence poses new threats to forensic analysts since malicious video alterations are more difficult to be detected. As a matter of fact, developing video analysis softwares that are able to expose video tampering is nowadays a crucial issue in many security and news-related applications, e.g., combating fake news, authenticating evidences.

In order to solve this problem, forensic researchers put a huge effort toward the development of video forensic solutions in the last few years [1]. In particular, several tasks of interest were identified and addressed, such as video device identification [2, 3], local tampering detection and localization [4, 5, 6, 7], physical inconsistencies detection [8], computer-generated video identification [8], video recapture understanding [9, 10], frame addition and removal analysis [11], detection of temporal interpolation [12], fake bitrate detection [13], video codec identification [14], and multiple compression detection [15].

Despite these solutions prove interesting, the achieved accuracy tends to decrease in the case of strong video compression [16]. Indeed, differently from standard image coding schemes, video codecs perform a complex set of operations to reduce the required bitrate. As a side effect, many traces exploited by forensic algorithms to detect processing operations are removed at the encoding stage. In real-world scenarios, video sequences are often distributed in compressed format due to storage limitations or bandwidth constraints. As a matter of fact, the loss of such revealing traces is an actual and frequent threat to forensic analyses.

In this paper, we explore the possibility of leveraging video codec traces as an asset for forensic purposes. Specifically, we focus on the detection of video temporal splicing. Every time different videos are temporally concatenated (e.g., to create a compilation, but also to add or substitute some frames), the original videos were seldom encoded with the exact same codecs or qualities. Therefore, it is possible to exploit coding traces inconsistencies on a frame-by-frame level, in order to detect possible temporal splicing and localize the splicing point over time.

To develop the proposed system, we leverage feature learning capability expressed by Convolutional Neural Networks (CNNs), which have recently shown very accurate results in many multimedia forensic tasks [17, 18, 19, 20]. Specifically, we train two different CNNs to extract video coding scheme and estimate the quality of video frames, then we search for CNN-extracted feature inconsistencies in the time domain. If inconsistent codecs and/or quality information are detected, the video is marked as a temporal splicing composition and the splicing point in time domain is finally localized.

To validate the proposed solution, we make use of a dataset containing sequences at different resolutions (i.e., CIF, 4CIF, PAL and 720p) encoded with four commonly used video codecs (i.e., MPEG2, MPEG4, H264 and H265) at different qualities (i.e., different constant and variable bitrates, as well as constant quantization parameters). Results show that it is possible to extract codec and quality information from video frames in a blind fashion through CNNs. Moreover, the developed splicing detection system can be generalized to those scenarios where training and test sets mismatch

This work has been partially supported by the University of Padova project Phylo4n6 prot. BIRD165882/16. This material is based on research sponsored by DARPA and Air Force Research Laboratory (AFRL) under agreement number FA8750-16-2-0173. 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.

in terms of video sequences, resolutions, and codec implementations.

2. PROBLEM FORMULATION

Let us denote with $\mathbf{X} = [\mathbf{X}(0), \mathbf{X}(1), ..., \mathbf{X}(N-1)]$ a video sequence composed by N frames $\mathbf{X}(n)$. Two video sequences $\mathbf{X}_i$ and $\mathbf{X}_j$ can be concatenated in time obtaining the spliced video $\mathbf{X}_{i,j} = [\mathbf{X}_i, \mathbf{X}_j] = [\mathbf{X}_i(0), ..., \mathbf{X}_i(N_i - 1), \mathbf{X}_j(0), ..., \mathbf{X}_j(N_j - 1)]$. The video compilation $\mathbf{X}_{i,j}$ is characterized by a splicing point at frame number $n = N_i - 1$, meaning that frames $\mathbf{X}_{i,j}(N_i - 1)$ and $\mathbf{X}_{i,j}(N_i)$ originally belong to two different video shots (i.e., $\mathbf{X}_i$ and $\mathbf{X}_j$, respectively).

In this work we propose a solution to the video splicing detection problem. This consists in detecting whether a generic video sequence under analysis is a composition of at least two shots (i.e., as $\mathbf{X}_{i,j}$), or it is a single original video (i.e., as $\mathbf{X}_i$ or $\mathbf{X}_j$), based only on pixel level analysis (i.e., not exploiting the bitstream or additional metadata). Moreover, we propose a solution to video splicing localization problem. This means being able to correctly identify the splicing point (i.e., the frame at which the splicing begins) in a video composition (i.e., frame at position $n = N_i - 1$ as the splicing point of $\mathbf{X}_{i,j}$).

Without loss of generality, in this work we consider spliced videos composed by only two shots (since it can be easily extended iterating the procedure). Additionally, we consider the case of compilations obtained by splicing shots encoded with different codecs and/or different quality parameters. This is the case of video compilations obtained with shots coming from different devices, different broadcasting sources, downloaded from different social media, as well as shots compressed several times due to post-processing operations (i.e., multiple compression can decrease quality). As, in a real-world case, videos are typically encoded again after splicing (i.e., videos are not distributed in raw format), we consider that all spliced videos are re-encoded.

3. PROPOSED SYSTEM

Given a video $\mathbf{X}$ under analysis, the proposed system for video splicing detection and localization can be synthesized by the following passages:

- A CNN trained to identify codec-related information extracts a feature vector $\mathbf{f}_{\text{C}}(n)$ from each frame $\mathbf{X}(n)$.
- A CNN trained to infer the compression quality level extracts a feature vector $\mathbf{f}_{\text{Q}}(n)$ from each frame $\mathbf{X}(n)$.
- Features $\mathbf{f}_{\text{C}}(n)$ and $\mathbf{f}_{\text{Q}}(n)$ are concatenated into the vector $\mathbf{f}_{\text{CQ}}(n)$ for each frame.
- Inconsistencies between adjacent feature vectors $\mathbf{f}_{\text{CQ}}(n)$ and $\mathbf{f}_{\text{CQ}}(n+1)$ are exploited to detect and localize splicing.

In the following, we report a detailed description of each step.

Video Codec CNN. Each video frame $\mathbf{X}(n)$ is split into non-overlapping 64×64 color patches $\mathbf{X}^p(n),\ p \in [0, P-1]$, where the number of patches P is bound by video resolution. Each patch is fed to a CNN tailored to solve a four-class classification problem:, i.e., detecting whether each patch comes from a video encoded using MPEG2, MPEG4, H264, or H265. Concerning the adopted network architecture, we empirically noticed a benefit in using a fully convolutional approach. To this purpose we replaced pooling layers with convolutional layers with stride 2 (i.e., filters are convolved moving them of two pixels per direction every time, resulting in a factor 2 downsampling). Specifically, the adopted CNN structure is the following one:

- Two convolutional layers with 32 filters of size 5 and stride 1.
- One convolutional layer with 32 filters of size 2 and stride 2 followed by SELU activation.
- Two convolutional layers with 48 filters of size 4 and stride 1.
- One convolutional layer with 48 filters of size 2 and stride 2 followed by SELU activation.
- Two convolutional layers with 64 filters of size 4 and stride 1.
- One convolutional layer with 64 filters of size 2 and stride 2 followed by SELU activation.
- One convolutional layer with 128 filters of size 3 and stride 1.
- One fully connected layer with 128 output neurons, followed by SELU activation.
- One fully connected layer with 4 output neurons, followed by Softmax activation.

The amount of trainable parameters is 325.140, thus making the network deep (i.e., 12 layers) but fast to train and deploy.

For each patch $\mathbf{X}^p(n)$, the network's output is a four-element feature vector

$$\mathbf{f}_{\text{C}}^p(n) = [f_{\text{H264}}^p(n),\ f_{\text{H265}}^p(n),\ f_{\text{MPEG2}}^p(n),\ f_{\text{MPEG4}}^p(n)], \quad (1)$$

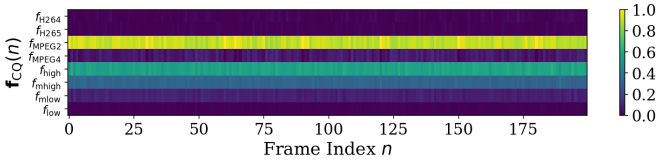
where each element represents the likelihood of the p-th patch from the n-th frame being encoded with one of the four considered codecs. Notice that, due to final Softmax activation, features vectors are non-negative and sum to one, thus being naturally normalized. The final frame-level codec feature vector $\mathbf{f}_{\text{C}}(n)$ is obtained by averaging patches' feature vectors as

$$\mathbf{f}_{\text{C}}(n) = \frac{1}{P} \sum_{p=0}^{P-1} \mathbf{f}_{\text{C}}^p(n), \quad (2)$$

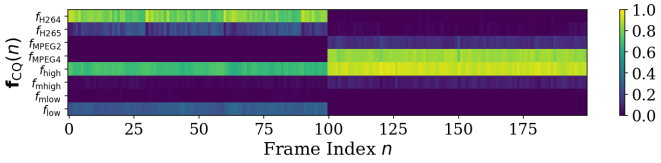
where all operations are performed in an element-wise fashion. This feature vector can be interpreted in different ways. On the one hand, we can consider each element as the likelihood of frame $\mathbf{X}(n)$ being encoded with a given codec in the set of considered four ones. On the other hand, we can simply interpret the distribution of the four likelihoods $f_{\text{H264}}(n)$, $f_{\text{H265}}(n)$, $f_{\text{MPEG2}}(n)$ and $f_{\text{MPEG4}}(n)$ as a general descriptor capturing codec traces. Indeed, for splicing detection and localization, we are not required to exactly detect the used codec for each frame, but we are more interested in observing some sort of codec incoherency over time.

Video Quality CNN. As for the previous step, each video frame $\mathbf{X}(n)$ is split into non-overlapping 64×64 color patches $\mathbf{X}^p(n)$. Each patch is additionally processed using the denosing algorithm presented in [21] to extract patch noises $\mathbf{W}^p(n)$. Noises are fed to a CNN trained to solve a four-class classification problem, i.e. to detect whether the patch comes from a frame encoded with low (low), medium-low (m-low), medium-high (m-high) or high (high) quality. In this case, we resorted to a more standard architecture similar to the one proposed in [22]:

- One convolutional layer with 32 filters of size 3 and stride 1, followed by Batch Normalization, ReLU activation, and Max Pooling of size 2 and stride 2.
- One convolutional layer with 64 filters of size 3 and stride 1, followed by Batch Normalization, ReLU activation, and Max Pooling of size 2 and stride 2.

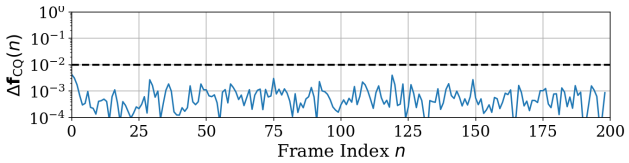


(a) Pristine

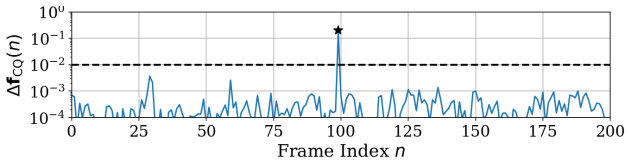


(b) Composition

Fig. 1. Feature vector for each frame of a pristine video (a) and a video composition (b). Composition codec changes after 100 frames.



(a) Pristine



(b) Composition

Fig. 2. MSE between adjacent feature vectors for a pristine video (a) and a video composition (b). Composition codec changes after 100 frames as denoted by the star. Videos are the same used for Fig. 1.

- One convolutional layer with 96 filters of size 3 and stride 1, followed by Batch Normalization, ReLU activation, and Max Pooling of size 2 and stride 2.
- One convolutional layer with 128 filters of size 3 and stride 1, followed by Batch Normalization, ReLU activation, and Max Pooling of size 2 and stride 2.
- One convolutional layer with 128 filters of size 3 and stride 1, followed by Batch Normalization, ReLU activation, and Max Pooling of size 2 and stride 2.
- One fully connected layer with 128 output neurons, followed by Dropout with probability 0.5.
- One fully connected layer with 4 output neurons, followed by Softmax activation.

For each patch noise $\mathbf{W}^p(n)$, the network's output is a four-element feature vector

$$\mathbf{f}_{\mathrm{Q}}^p(n) = [f_{\text{low}}^p(n),\ f_{\text{m-low}}^p(n),\ f_{\text{m-high}}^p(n),\ f_{\text{high}}^p(n)], \tag{3}$$

where each element represents the likelihood of the patch coming from a frame encoded with different quality in a set of four possible choices. Also in this case, we obtain the final frame descriptor $\mathbf{f}_{\mathrm{Q}}(n)$ by averaging feature vectors coming from all the patches extracted from the same frame

$$\mathbf{f}_{\mathrm{Q}}(n) = \frac{1}{P}\sum_{p=0}^{P-1}\mathbf{f}_{\mathrm{Q}}^p(n), \tag{4}$$

where all operations are performed element-wise. In our scenario, we can interpret this feature vector as a compact descriptor of frame coding quality. Due to Softmax normalization, also this feature vector is bound to be non-negative and all elements sum to one.

Splicing Detection and Localization. After feature vectors $\mathbf{f}_{\mathrm{C}}(n)$ and $\mathbf{f}_{\mathrm{Q}}(n)$ are extracted from a frame, we concatenate them into a single eight-element feature vector $\mathbf{f}_{\mathrm{CQ}}(n) = [\mathbf{f}_{\mathrm{C}}(n),\ \mathbf{f}_{\mathrm{Q}}(n)]$. Fig. 1a shows an example of $\mathbf{f}_{\mathrm{CQ}}(n)$ for a video composed by 200 original frames encoded with high-quality MPEG2, whereas Fig. 1b shows an example of video composed by 100 frames encoded with high-quality H264 spliced with 100 frames encoded with high-quality MPEG4. In the second example, it is possible to observe an evident feature vector inconsistency at frame number 100.

To automatically detect this inconsistency, thus detect splicing, our method works as follows. We compute the mean squared error (MSE) between feature vectors belonging to adjacent frames

$$\Delta\mathbf{f}_{\mathrm{CQ}}(n) = \mathrm{MSE}(\mathbf{f}_{\mathrm{CQ}}(n),\ \mathbf{f}_{\mathrm{CQ}}(n+1)). \tag{5}$$

We then compare the maximum value of $\Delta\mathbf{f}_{\mathrm{CQ}}(n)$ with a threshold Γ. If $\max(\Delta\mathbf{f}_{\mathrm{CQ}}(n)) > \Gamma$, then the video is detected as spliced. In this case, the maximum $\Delta\mathbf{f}_{\mathrm{CQ}}(n)$ position represents the splicing point

$$\hat{n} = \arg\max_{n}(\Delta\mathbf{f}_{\mathrm{CQ}}(n)). \tag{6}$$

Fig. 2 shows $\Delta\mathbf{f}_{\mathrm{CQ}}(n)$ referred to videos used for the example in Fig. 1 on a log-scale. It is possible to observe that, in case of splicing (i.e., Fig. 2b), the splicing point becomes evident.

4. SIMULATIONS AND RESULTS

In this section we report all the details about the performed simulations in terms of dataset generation and training protocols. Then we report all the achieved results, separately evaluating each step of the proposed method.

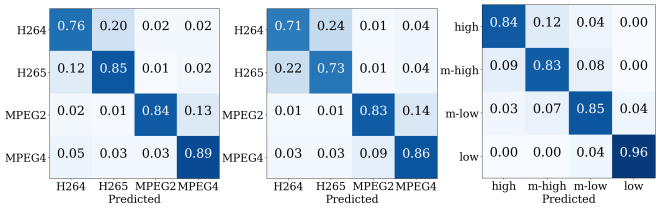
Datasets. In order to train the two different CNNs and test the whole system, we prepared different datasets by compressing a set of diverse training sequences with different coding set-up, frame resolutions, and codec types [1]. This is essential to prove CNN generalization capability and evaluate the whole pipeline.

To train the video codec CNN, we built dataset $\mathcal{D}_{\text{train}}^{\text{HR}}$ composed by 300 videos at high resolution. We started from five uncompressed video sequences, namely: *ducks take off* (720p), *stockholm* (720p), *ice* (4CIF), *harbour* (4CIF), *parkrun* (720p). Each sequence has been encoded using FFmpeg to obtain 60 different versions combining codecs and qualities. As codecs we considered MPEG2, MPEG4, H264, H265. As quality, we considered: fixed quantization parameter (QP) ranging from 1 to 10; constant bitrate set to 2 Mb/s, 4 Mb/s and 6 Mb/s; variable bitrate set to 2 Mb/s, 4 Mb/s and 6 Mb/s. As group of pictures (GOP) we used 30 frames.

To validate the video codec CNN (i.e., select the trained CNN model), we built dataset $\mathcal{D}_{\text{val}}^{\text{HR}}$ composed by 300 videos at high resolution. This is obtained following the same procedure of $\mathcal{D}_{\text{train}}^{\text{HR}}$, starting from other original sequences: *park joy* (720p), *parkrun* (720p), *shields* (720p), *soccer* (4CIF), and *stockholm* (720p).

To test the video codec CNN on a completely unrelated set, we built dataset $\mathcal{D}_{\text{test}}^{\text{LR}}$ composed by 1.672 videos at low resolution

[1] Original videos at: `https://media.xiph.org/video/derf/`



(a) Validation on $\mathcal{D}_{\text{val}}^{\text{HR}}$ (b) Validationon $\mathcal{D}_{\text{test}}^{\text{LR}}$ (c) Validation on $\mathcal{D}_{\text{quality}}^{\text{SR}}$

Fig. 3. Video codec and coding quality identification confusion matrices. Results are shown at patch level.

(i.e., CIF). We started from 19 sequences at CIF resolution (*akiyo*, *crew*, *mother*, *soccer*, *bridgeclose*, *flower*, *news*, *table*, *city*, *foreman*, *paris*, *tempete*, *coastguard*, *hall*, *salesman*, *waterfall*, *container*, *mobile*, *sign irene*), and encoded them using FFmpeg mixing codecs and qualities. As codecs we considered MPEG2, MPEG4, H264, H265. As quality, we considered: fixed quantization parameter (QP) ranging from 1 to 32 with step 2; constant bitrate set to 500 Kb/s, 1 Mb/s and 2 Mb/s; variable bitrate set to 500 Kb/s, 1 Mb/s and 2 Mb/s. As group of pictures (GOP) we used 10 frames.

To train and validate the coding quality CNN, we prepared dataset $\mathcal{D}_{\text{quality}}^{\text{SR}}$ of 80 videos at PAL standard resolution. We started from five uncompressed sequences, namely: *crew*, *ducks take off*, *harbour*, *ice* and *soccer*. We encoded them using four different versions of H264 reference software (i.e., HM8.0, HM9.2, HM10.1 and HM12) in order to accurately control the quantization parameter set-up. We set low, m-low, m-high and high qualities to fixed QP equal to 5, 10, 15 and 20. As GOP size we used 10 frames. The 75% of the extracted frames have been used for training, the rest for validation.

Finally, to test the whole system, we prepared dataset $\mathcal{D}_{\text{splice}}$ composed by 100 spliced and 100 original videos. We started from five uncompressed sequences at 720p resolution never used in other sets: *four people*, *in to tree*, *johnny*, *kristen and sara*, *old town cross*. Using FFmpeg, we encoded them using the four codecs MPEG2, MPEG4, H264, H265, with GOP set to 30, and fixed QP ranging from 3 to 20. A random set of these sequences (trimmed to 200 frames) has been used as pristine. Another random set has been used to create 100 compositions of 200 frames each, made by splicing together two different portions of the same video encoded with different codecs and / or parameters. Composition are then re-encoded using high-quality H264.

Notice that spliced videos do not present any scene change, as the same video was used for the head and tail. Simply the first 100 frames are encoded differently with respect to the other ones. This is important to assess that our algorithm does not simply detect scene changes, but actually localizes coding changes.

Training Strategy. Both CNNs have been trained using the same methodology. Categorical crossentropy has been used as loss function. Adam optimizer with standard parameters and learning rate has been used for loss minimization. The best model has been selected as the one minimizing loss on the validation set over 100 epochs.

Video Codec Identification Results. Fig. 3a and Fig. 3b show confusion matrices obtained in terms of codec identification by the video codec CNN at patch-level on validation dataset $\mathcal{D}_{\text{val}}^{\text{HR}}$ and test dataset $\mathcal{D}_{\text{test}}^{\text{LR}}$, respectively. It is interesting to notice how results are in line on both datasets, despite sequences have been generated starting from content at different resolutions and using different bitrates. Moreover, we can observe that H264 tends to be confused with H265, while MPEG2 is confused more with MPEG4. This is due to natural similarities among these families of codecs.

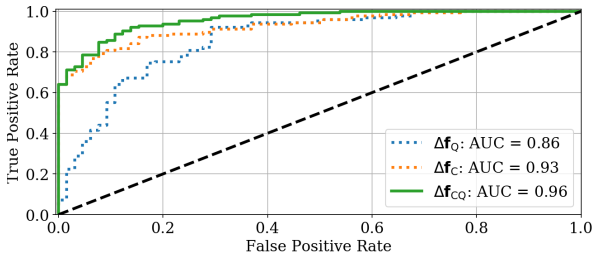


Fig. 4. Temporal splicing detection ROC curve using different sets of features.

Table 1. Perfect detection rate and mean absolute error in frames using different approaches.

Method	Perfect Detection Rate	Mean Absolute Error
$\Delta\mathbf{f}_{\text{Q}}$	0.520	22.25 frames
$\Delta\mathbf{f}_{\text{C}}$	0.736	12.94 frames
$\Delta\mathbf{f}_{\text{CQ}}$	0.856	7.12 frames

Quality Identification Results. Fig. 3c shows the confusion matrix obtained in terms of video coding quality detection on the validation set $\mathcal{D}_{\text{quality}}^{\text{SR}}$. Also in this case we can notice good average performance of the CNN in distinguishing the four selected quality levels for each patch.

Temporal Splicing Detection and Localization Results. In order to evaluate the proposed splicing detection pipeline, we computed Receiver Operating Characteristic (ROC) curves by comparing $\max(\Delta\mathbf{f}_{\text{CQ}}(n))$ value for each sequence in $\mathcal{D}_{\text{splice}}$ with a variable threshold Γ. Fig. 4 shows the system performance using different sets of features. Using only quality-based features (i.e., $\Delta\mathbf{f}_{\text{Q}}$), the area under the curve is 0.86. By using only features coming from the codec-based CNN (i.e., $\Delta\mathbf{f}_{\text{C}}$), AUC increases to 0.93. Anyway, the best result is obtained when all features are jointly used (i.e., $\Delta\mathbf{f}_{\text{CQ}}$), providing an AUC of 0.96.

In terms of localization, we report in Table 1 the perfect detection rate (i.e., the percentage of times we estimate the exact splicing point), and the mean absolute error (i.e., how far on average is the estimated splicing point from the true one). Also in this case, it is possible to notice that the best results are obtained using all features jointly (i.e., 85.6% of perfect detection, and average error of less than 8 frames).

5. CONCLUSIONS

In this paper we presented a video temporal splicing localization and detection system exploiting only traces left by video coding. To capture these traces, we separately train two CNNs: one devoted to capture characteristics of the used video codec; one devoted to capture characteristics of the used coding quality. The CNNs and the whole system are validated on different dataset to avoid overfitting problems and assess generalization capability.

Being able to capture coding traces on small frame patches paves the way to the possibility of extending the proposed solution to local tampering detection. Future research will be devoted to study coding footprints locally on each frame, in order to detect possible copy-move and other types of forgeries.

6. REFERENCES

[1] S. Milani, M. Fontani, P. Bestagini, M. Barni, A. Piva, M. Tagliasacchi, and S. Tubaro, "An overview on video forensics," *APSIPA Transactions on Signal and Information Processing*, vol. 1, pp. e2, 2012.

[2] M. Chen, J. Fridrich, M. Goljan, and J. Lukas, "Source digital camcorder identification using sensor photo-response nonuniformity," in *SPIE Electronic Imaging (EI)*, 2007.

[3] S. Bayram, H. T. Sencar, and N. Memon, "Video copy detection based on source device characteristics: A complementary approach to content-based methods," in *ACM International Conference on Multimedia Information Retrieval*, 2008.

[4] C.-C. Hsu, T.-Y. Hung, C.-W. Lin, and C.-T. Hsu, "Video forgery detection using correlation of noise residue," in *IEEE Workshop on Multimedia Signal Processing (MMSP)*, 2008.

[5] P. Bestagini, S. Milani, M. Tagliasacchi, and S. Tubaro, "Local tampering detection in video sequences," in *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2013.

[6] L. D'Amiano, D. Cozzolino, G. Poggi, and L. Verdoliva, "Video forgery detection and localization based on 3D patchmatch," in *IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, 2015.

[7] D. D'Avino, D. Cozzolino, G. Poggi, and L. Verdoliva, "Autoencoder with recurrent neural networks for video forgery detection," in *IS&T Electronic Imaging (EI)*, 2017.

[8] V. Conotter, J. O'Brien, and H. Farid, "Exposing digital forgeries in ballistic motion," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 7, pp. 283–296, 2012.

[9] M. Visentini-Scarzanella and P. L. Dragotti, "Video jitter analysis for automatic bootleg detection," in *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2012.

[10] A. Hajj-Ahmad, S. Baudry, B. Chupeau, G. Dorr, and M. Wu, "Flicker forensics for camcorder piracy," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 12, pp. 89–100, 2017.

[11] M. Stamm, W. Lin, and K. Liu, "Temporal forensics and anti-forensics for motion compensated video," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 7, pp. 1315 –1329, 2012.

[12] P. Bestagini, S. Battaglia, S. Milani, M. Tagliasacchi, and S. Tubaro, "Detection of temporal interpolation in video sequences," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[13] S. Bian, W. Luo, and J. Huang, "Exposing fake bit rate videos and estimating original bit rates," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 24, pp. 2144–2154, 2014.

[14] P. Bestagini, S. Milani, M. Tagliasacchi, and S. Tubaro, "Codec and GOP identification in double compressed videos," *IEEE Transactions on Image Processing (TIP)*, vol. 25, pp. 2298–2310, 2016.

[15] S. Milani, P. Bestagini, M. Tagliasacchi, and S. Tubaro, "Multiple compression detection for video sequences," in *IEEE International Workshop on Multimedia Signal Processing (MMSP)*, 2012.

[16] W.-H. Chuang, H. Su, and M. Wu, "Exploring compression effects for improved source camera identification using strongly compressed video," in *IEEE International Conference on Image Processing (ICIP)*, 2011.

[17] J. Chen, X. Kang, Y. Liu, and Z. J. Wang, "Median Filtering Forensics Based on Convolutional Neural Networks," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1849–1853, November 2015.

[18] A. Tuama, F. Comby, and M. Chaumont, "Camera model identification with the use of deep convolutional neural networks," *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2016.

[19] B. Bayar and M. C. Stamm, "A deep learning approach to universal image manipulation detection using a new convolutional layer," *ACM Workshop on Information Hiding and Multimedia Security (IHMMSEC)*, 2016.

[20] L. Bondi, S. Lameri, D. Güera, P. Bestagini, E. J. Delp, and S. Tubaro, "Tampering detection and localization through clustering of camera-based cnn features," *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017.

[21] M. K. Mihcak, I. Kozintsev, K. Ramchandran, and P. Moulin, "Low-complexity image denoising based on statistical modeling of wavelet coefficients," *IEEE Signal Processing Letters (SPL)*, vol. 6, pp. 300–303, 1999.

[22] L. Bondi, L. Baroffio, D. Güera, P. Bestagini, E. J. Delp, and S. Tubaro, "First steps toward camera model identification with convolutional neural networks," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 259–263, March 2017.