International Journal of Scientific Research and Engineering Development-– Volume 7 Issue 2, Mar-Apr 2024

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Video Surveillance Anomaly Detection Based on a 2D Convolutional Neural Networks Combined with Recurrent Neural Networks

Konan Atchelo Desire Bienvenu*, Goore Bi Dje Jean Romain Desire**

Computer Science, Nanjing University of Information Science and Technology, Nanjing

Email: atchelochina@gmail.com

Computer Science, Nanjing University of Information Science and Technology, Nanjing

Email: desiregoore5@gmail.com

----------**********----------

Abstract:

This study addresses the challenges posed by automated video surveillance, focusing specifically on crowd behavior analysis. Moreover, it explores the utilization of spatio-temporal features for action anomaly detection, aiming to enhance surveillance system capabilities. In our experimental endeavors, we propose a novel model leveraging a combination of 2D CNNs and recurrent neural networks (RNNs) for improved processing of video sequences. Our hypothesis posits that this amalgamation preserves the semantic structure of video data more effectively, leading to more meaningful information extraction.Through rigorous experimentation, our model demonstrates superior classification performance compared to the original approach, despite being trained on a smaller dataset. This substantiates our hypothesis and underscores the efficacy of our proposed method in action anomaly detection tasks. Overall, this research contributes to advancing the field of automatic video surveillance, highlighting the potential benefits of integrating convolutional and recurrent networks for complex data analysis.

Keywords **—Deep learning, crowd analysis, anomaly detection, video-surveillance, Convolutional Neural Network (CNN), recurrent neural networks (RNNs).**

----------**********----------

I. INTRODUCTION

We will study the use of spatio-temporal features for the detection of anomalous actions in video. Specifically, we will take as a starting point the model proposed in the paper Real-World Anomaly Detection in Surveillance Videos [1], and we will make a proposal for improvement. Our approach consists of changing the originally employed feature extractor, which is based on the use of 3D convolutions [2], for one based on classical 2D convolutions followed by a recurrent stage.

Our hypothesis argues that, although 3D convolutions are designed to extract information from video fragments, they are not capable of correctly capturing long-term temporal dependence. The lack of capability in this regard results in an overall model deterioration, which can be remedied with a more powerful feature extractor that makes use of both convolutional layers to extract spatial information and recurrent layers to learn the temporal sequence.

Next sections will detail the experimentation carried out, describing the data set used, the original model, and the proposed modifications.

II. SPATIO-TEMPORAL FEATURES ANALYSIS

A. Dataset used: UCf101-Crime Dataset

The data set with which we worked was proposed by the authors of the article under study. This dataset is composed of video sequences extracted from video surveillance cameras. As can be seen on the project website, the database is composed of

1900 videos of varying length, with an average of 7247 frames. We are therefore talking about videos of an acceptable averaging several minutes in length. In total, 128 hours of video are available, and 13 different classes of anomalies are included, as well as videos that are considered normal. Normal videos are considered to be those in which no anomalous behaviour appears. Some of the anomaly classes included in the set are abuse, theft, burglary, breaking and entering, shooting, store robbery, fighting or explosions. One particularity of the dataset is the great diversity of frames in which the videos are collected. Figure 1 shows different frames extracted from normal videos. As can be seen, we have videos taken by indoor and outdoor cameras, both in daylight and at night, and with different camera angles. This limits the possibility of making assumptions about the data, and forces us to build a generic model, able to work in different situations.

(a) Indoor at eye level (b) Daytime exterior

(c) Interior from above (d) Exterior at night

Fig. 1 Examples of normal frames from the UCF-Crime dataset

The second problem is to correctly assign the class to which the videos labelled as anomalous belong. In our case, we will focus on solving the first problem. The authors of the original model give much more importance to the first part than to the second, and therefore we have focused on their analysis, ignoring the second part. Therefore, for us there will be no distinction between the different kinds of anomalies, so from this point on we only distinguish between normal and abnormal videos. The data set is divided into training and test subsets. For the training set, 800 normal videos and 810 abnormal videos are available. For the test data set, 290 videos (150 normal, 140 abnormal) are available.

The main peculiarity of the dataset is that the training subset is weakly labelled. This means that, although the objective to be addressed is to temporally locate when an anomaly occurs, the labels available in the training set only mark that the video is anomalous, not where the anomaly occurs. Therefore, instead of having for each video a binary value per frame indicating whether or not an anomaly is present in that frame, what we have is a single binary label for the entire video.

In the test set, however, the labelling is at the frame level, and therefore we will have to learn to temporarily locate the anomaly in a longer fragment. This makes it necessary to design a specific learning system to learn at the frame level when such information is not available.

On the labels of the test set, we observe that we are faced with a strongly unbalanced problem. In total, we have 1027477 frames labelled as normal frames, while there are only 84331 anomalous frames. Surprisingly, despite having the sets practically balanced in terms of anomalous and normal frames, when we perform the calculation at the frame level, there is a very significant unbalance. This is because, in reality, the videos that present an anomaly concentrate the anomaly in a few seconds, and most of the video time is composed of normal frames.

B. Original model

This architecture consists of a feature extractor based on convolutional neural networks in three dimensions, followed by a fully connected network that performs the final classification. In addition, the main contribution of the model is a loss function that allows learning frame-level labelling despite training with video-level labelling. This function solves the labelling peculiarity of the training dataset in a weak way. In the following sections wewill describe in detail each of the parts of the network, as well as the proposed loss function.

1) Feature extractor: C3D

The feature extractor used in the original model is the model known as C3D [2]. The particularity of

this model is that instead of using two dimensional convolutions, which are typically used in image processing, three-dimensional convolutions are used. The difference is that the two dimensional convolution only shifts the kernel along the width and height of the image. The convolution kernel is therefore a matrix, and the convolution output is in two dimensions.When a 3D convolution is applied, the temporal dimension is also taken into account. In this case, the kernels are rank 3 tensors, and the convolution does not use a single frame, but takes into account several consecutive frames. In this way, not only spatial information is collected (in the height and width dimensions), but also temporal information (since the time dimension is involved). Thus, the output is in three dimensions, instead of two. The proposed extractor therefore consists of a series of 3D convolutions that take a video as input and extract features from it.

According to the authors of the model, the output of the first fully connected layer is the one to be used as the representation of the fragment of video. The model input is a 16-frame video of size 112 × 112, and returns as output a 4096-element descriptor of that video.The feature extractor used in the work is previously trained on a dataset whose objective is the classification of behaviours. This large dataset is often used as a starting point for training models dedicated to feature extraction in video, similar to the use of ImageNet for images.

Since the neural networks used in this type of work usually require large datasets to be trained, the sets used are not usually large enough for the learning to be sufficiently rich. For this reason, a previous training stage is usually performed on a large dataset, with which generic features are learned, and then the model is refined on the dataset being worked on. This technique is known as transfer learning.

Specifically, in the work we are analysing, the feature extractor is trained on the Sports-1M dataset [3]. That dataset consists of 1133158 videos, with a total of 487 distinct classes. The feature extractor is trained by placing a densely connected layer at the end of the previous architecture, with as many neurons as classes in the dataset (487 in this case), and training the entire network to solve the classification task. Once the network is trained to solve the problem, because the behaviours present in the dataset are very varied, the layers of the network are able to extract very diverse information from the input videos. To build the final feature extractor, the last classification layer is removed and the weights learned for the rest of the layers are kept. This last technique is what is known as freezing the network.

After this training, we obtain a model capable of summarizing the information of a 16-frame video into a vector of 4096 components. This model will be used to summarize the information of each video in the set into 16-frame fragments. In this way, for each video, we will have a variable number of extracted descriptors. In the next section we will discuss how these descriptors are converted into a fixed size representation of the video, and how this information can be used to train the final classifier. To do this, we will need to define a cost function that allows us to learn at the frame level (or a few frames), despite having labels only at the video level.

2) *Multi-instance learning*

In this section we present the learning policy designed in the article for training the model. As discussed above, we have labels at the video level, but not at the frame level, which is the task we really want to solve.

The training approach is based on what is known as multi-instance learning. Instead of receiving a label for each item in the dataset, a bag of uniquely labelled items is received. In the case of binary classification, a bag will be labelled as negative if all the elements in the bag are negative elements. On the other hand, a bag will be labelled positive if at least one of its elements is positive. This translates very simply to our scenario. In our case, each video will represent a single bag, labelled negative if it comes from a normal video (where no anomalies occur), and positive if it comes from an abnormal video (we know that an anomaly occurs in that video, but not where). To achieve a fixed-size representation for each video, the videos are divided into 32 temporal segments, resulting in 32 elements in each bag. To define the occupancy of each segment, the previously processed 16-frame fragments are distributed equispaced, so that in

each segment there are the same number of fragments, all of them consecutive (or the most equal distribution possible, if the division is not exact). The representation of the segment is the average of the representations of the fragments that form it. Thus, for each video, 32 different elements are obtained, which will form the bag in question.

Now, we have to define the loss function to train with the model, given the bags we have obtained. In a normal context, if we had examples of anomalous V_a and normal V_n videos labeled complete, we would want to get a model whose output was a score anomaly, such that however, what we have are bags of examples, rather than complete videos.

$$f(V_a) > f(V_n)$$

In these bags we know that there are positive examples, but we do not know what they are. Given two bags of examples, one coming from a normal video B_n and one coming from an anomalous video B_a, we will want to follow the score of the highest scoring segment of the bag The normal video (that piece of video that being normal is difficult to classify) is higher than the highest scoring segment of the abnormal bag (that segment of the abnormal bag with a higher probability of being abnormal). In other words, our intention will be to force with this formulation, we want to ensure that fragments that actually contain an anomaly have a higher score than those that do not, but are difficult to classify examples.

$$\max_{i \in B_a} f\left(V_a^i\right) > \max_{i \in B_n} f\left(V_n^i\right)$$

Using a similar idea to the Hinge loss function, which is used in SVM training to maximize the classification margin, we define the loss function for model training as in addition, the above function we have ignored the temporal structure of the input video.

$$l(B_a, B_n) = \max\left(0{,}1 - \max_{i \in B_a} f\left(V_a^i\right) + \max_{i \in B_n} f\left(V_n^i\right)\right)$$

In a video with anomalies, we have that most of the time we encounter normal frames. For this reason, we want most of the predictions we make to be close to zero. In addition, we are interested in preventing the change in the anomaly score from being too abrupt, so we want the increase in the anomaly score between one segment and the next not to be too strong. This leads us to add two regularization terms to the cost function, which are responsible for controlling the two quantities we have mentioned. With such a modification, the final loss function becomes the term accompanying parameter λ_1 refers to the temporal regularization term, and the term accompanying parameter λ_2 refers to the sparse regularization the values λ_1 and λ_2 can be adjusted to give more or less importance to the regularization terms.

$$\begin{aligned} l(B_a, B_n) = \max\Bigg(0{,}1 \\ - \max_{i \in B_a} f\left(V_a^i\right) + \max_{i \in B_n} f\left(V_n^i\right)\Bigg) \\ + \lambda_1 \sum_{i=0}^{n-1} \left(f\left(V_a^i\right) - f\left(V_a^{i+1}\right)\right) \\ + \lambda_2 \sum_{i=0}^{n} f\left(V_a^i\right) \end{aligned}$$

Once we have seen the training policy of the classifier model, we move on to give the complete description of the original model.

*3) **Complete original model***

Once we have seen the feature extractor and the training policy, we move on to see how the complete original model is trained.

Figure 2 shows the complete structure of the original model. Given two videos from the dataset, one normal and one anomalous, its 32 temporal segments are extracted and the bags of examples are formed. The 32 segments are processed with the pre-trained feature extractor, and summary descriptors are extracted for each segment. These descriptors are classified using a fully connected network, the only output of which is a neuron. This neuron learns to return an anomaly score for each segment. When the anomaly scores for the positive and negative videos have been computed, the value of the loss function is calculated and the weights of the dense layers are updated using a gradient descent method. The feature extractor is pre-trained and frozen, so at full model training time its weights are not updated.

For each training step, 60 videos are randomly selected from the set in a balanced way (30 normal and 30 abnormal), and the gradient of the loss

function is calculated based on the classification of the 60 videos.

Once the model is trained with the previous system, in inference time the video to be classified is divided in the same way as in the previous system.

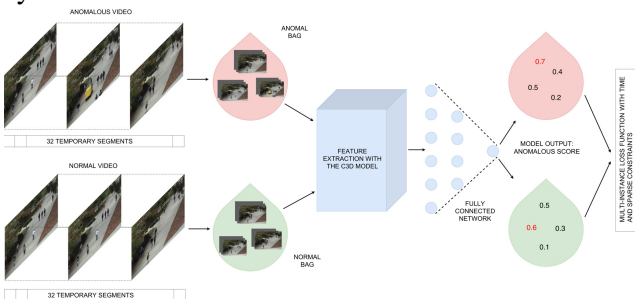


Fig. 2 Complete architecture of the original model

The anomaly score is calculated for each segment. To obtain a prediction for each frame, the prediction of the segments is interpolated. Having seen the original model in its entirety, we will now describe the proposed modification.

C. Proposed model improvement

The proposed improvement of the model consists of a modification of the feature extractor by a system that correctly captures the temporal information contained in the video fragments.

The main criticism that can be made of the previous model is that the feature extractor is not ideal. Although three-dimensional convolutions are models designed for feature extraction in video, their capacity to extract temporal information contained in a large number of consecutive frames is limited, since the receptive field of the network is limited by the size of the kernels used. In the case of the previous model, we work with cores of size 3, so we will extract information that is expressed in three consecutive frames, not at a greater distance. Thanks to the Pooling operation, in more advanced layers of the network, information is extracted from frames that are at a greater distance, but in our opinion the time dependence is underrepresented in these models.

Our proposal, therefore, tries to solve this problem by providing an extractor with higher quality spatio-temporal characteristics, which is described below.

4) Feature extractor: Xception-LSTM

As mentioned above, the proposal is based on modifying the feature extractor of the previous model, in such a way that an architecture capable of correctly capturing temporal information is used. For this purpose, we will make use of recurrent neural networks. This type of model differs from classical neural networks in that it connects the output of neurons of a layer with inputs from neurons within the same layer. Thus, their prediction depends not only on a single element (or a small set of elements) of the input vector, but also on the information received by the network in previous stages. In particular, we will use two dimensional convolutional networks to extract information from all the frames of the video, and we will create a time series with these descriptions. The time series will be used as input to a recurrent network that will learn the temporal pattern.

Specifically, we will use an LSTM architecture [4]. This architecture was proposed to solve the problem of forgetting that was detected in the first recurrent neural networks. This problem is that, although theoretically recurrent neural networks should be able to learn long-lasting temporal patterns, it was observed that when the information was sufficiently separated the effectiveness of these models deteriorated significantly. To mitigate this deterioration, what the LSTM neural network uses is an internal state that is transmitted between cells, in addition to the network output itself. In this way, the output of each neuron is partly separated from the internal state of the network that is transmitted to subsequent neurons. It was observed that this separation is beneficial and significantly improves the results obtained by the network. The basic operation of these neurons consists of combining the network memory (represented by the internal state and the previous output) with the current input by means of three different modules, known as gates. In the recurrent network we have connections between the neurons of the hidden layer, which does not occur in classical neural networks upward. Each neuron receives, in addition to the corresponding input, the previous internal state and output of the layer.

As in the original model, we will work with 16-frame video fragments. We will independently introduce the frames into an Xception model [5],

pre-trained on the ImageNet dataset [6] (this model is available for direct download in the Keras framework [7], which has been used for the implementation). The output of the penultimate layer of the network, which is a Global Average Pooling layer, is used as the descriptor of each frame. This layer summarizes all the information of a feature map by averaging the values of all the pixels that make it up. The output obtained is a vector of 2048 values.

Consequently, the frame representation of a video is a matrix of dimensions 16 * 2048. This representation is fed into an LSTM layer, which will consider the input as a time series of 16 time slots with 2048 features.

The output of such a layer will be a descriptor representing the spatiotemporal information contained in the 16-frame video fragment. Experiments have been carried out with different layer sizes. Specifically, descriptors of size 512, 768 and 1024 have been proposed, with better results being obtained the larger the descriptor size.

Given the lack of computational resources we have experienced when training the models, instead of having pre-trained the feature extractor on the Sports-1M set (on which the original extractor was pre-trained), we have pre-trained our extractor on the UCF-101 dataset [8]. That set has 13320 videos spread over 101 different classes. The duration of the videos is relatively short, ranging from just over a second to just over a minute for the longest videos.

As we can see, we are dealing with a dataset of a much smaller size than the one used by the authors of the original paper. This difference in size will mean that the features we learn with our feature extractor will be less rich and varied than those we could learn with a larger set, which may result in a loss of model performance. Nevertheless, we are talking about a data set of acceptable size for model pre-training.

As for the training policy, we will add to the feature extractor a block of densely connected layers at the end, which will perform the classification of the videos. To create the training examples, from the videos in the UCF-101 dataset we will select 16 equispaced frames within the whole video, and scale the frames to have the input size required by the Xception network (the input will have a final size of 16 × 299 × 299). The convolutional network weights will be frozen, to reduce the computational burden of training, since we consider the features learned in ImageNet to be of good quality for image classification. Therefore, we will only train the recurrent layer and the fully connected layers.

Once the complete training has been carried out, we remove the densely connected layers, and use the output of the recurrent layer as the descriptor of the video fragment. We thus have a 512, 768 or 1024 element descriptor for each 16-frame fragment. In Figure 3 we can see the complete feature extractor, ready for pre-training.

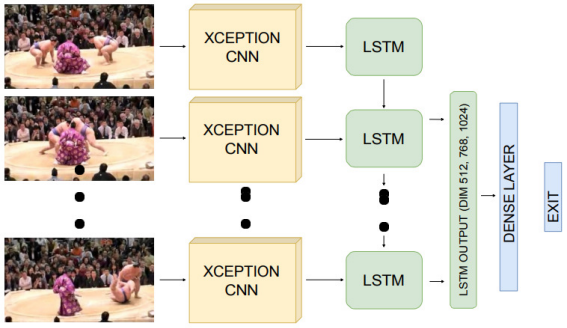


Fig. 3: Proposedfeature extractor, along with fully connected layers for training.

Once the model has been pre-trained on the UCF-101 set, the fully connected layers are removed. Therefore, our final feature extractor is composed of replicas of the Xception network for each of the input frames, and an LSTM network with output size 512, 768 or 1024, which will be what we will know as the video fragment descriptor.

III. EXPERIMENTATION AND RESULTS OBTAINED

A. Implementation aspects

The code developed for the work, which allows replicating the experimentation carried out, is available in the repository https://github.com/atchelodev/vs-anomaly-detection. The models provided are the ones for the experiment in which a 1024 size representation is used. The rest of the models have not been provided because the experimentation is similar in all cases, and we have preferred to include only the best of the models.

The code localized at https://github.com/ptirupat/AnomalyDetection_CVPR18 is used as the basis for the original implementation. An implementation of the model, as well as the compressed file of the original trained architecture, is provided in this repository. However, the code needed to retrain the system is not available. In addition, the author of the article provides the code at https://github.com/WaqasSultani/AnomalyDetectionCVPR2018.

In order to make the comparison between our model and the original more fair, we have chosen to repeat the experimentation carried out by the authors, in order to avoid possible differences in the training of the models. This is because we cannot guarantee that we are performing exactly the same steps as the original authors from the code they provide, since it is not complete. For example, in neither of the two repositories are the code for feature extraction is provided, so we cannot be sure that the results obtained by the original model from the features extracted by us are identical to those originally obtained. Nevertheless, we also include their results, which show that the model trained by us does not have exactly the same behaviour as theirs.

In this way, we will have two versions of the original model, one based on the trained model provided by the authors, and the other fully trained by us. As for the full implementation and training of the original model, the two sources above have been used to replicate the experimentation as reliably as possible. However, because much of the code was replicated between the two repositories, and some parts were inefficiently implemented, some changes were made to parts of the code while respecting its original operation. The original code of the article has not been used directly due to the impossibility of replicating some of the stages. For example, there are parts of the system written in MATLAB, or that require the use of external tools. The implementation we have carried out allows us to fully repeat the experimentation using Python exclusively, and without the need to resort to external sources.

The OpenCV library [9] has been used to process the data in video format. In particular, we have used the modules that allow the reading of videos from disk to memory. This library loads the videos as a list of numerical matrices, which represent each of the frames that make up the video sequence. For the handling of these matrices, we have made use of the NumPy numerical computation library [10, 11]. This package provides an interface to operate efficiently with structured collections of numbers (N-dimensional arrays). Because Python is a language oriented towards flexibility and ease of use, it has as a trade-off a significant loss of performance when performing arithmetic operations. For this reason, NumPy is an essential tool for the development of applications with high computational cost. In addition, for the effective management of annotation files, which are provided in CSV format, use has been made of Pandas [12]. This library allows the management of data structured in tables, using structures known as DataFrames.

For the implementation of neural network models, the Keraslibrary[7] has been used. This library provides a high-level interface for the implementation of deep learning models, using underneath the TensorFlow library [72]. The use of these two software packages greatly facilitates model development and validation, while offering some flexibility. This allows testing on architectures of some complexity without the need to write the entire system at a low level, but simply by combining neurons of various types organized in layers. In addition, the use of TensorFlow allows the execution of models on GPU architectures, instead of CPU. Because neural networks need to operate on high-dimensional matrices, running these models requires performing a very large number of arithmetic operations. The execution of these operations is computationally very expensive, and the ability to use the parallel processing of graphics cards greatly reduces model execution times.

As for the hardware architecture on which the models were run, the executions were carried out on a compute node with NVIDIA Tesla V100 graphics cards with 32 GB of graphics RAM. All models have been run using a single graphics unit at the same time.

B. Our experimentation

1) Xception-LSTM extractor training

The training of the feature extractor has been performed on a larger dataset, and the model obtained has been frozen and used to extract the video representations of our target set.

The training of the extractor has been performed on the UCF-101 dataset, as described above. To control the evolution of the model and check if it over-fits, the training and validation sets proposed by the authors of the data set as first partitions are used as training and validation sets. In order to make the comparison between models fair, the data set is divided into training and testing three times. In this way, the models are required to be trained and evaluated three times, thus reducing the possible influence of chance. In our case, we will use the first of the three splits to train our extractor. Using the test partition as the validation set, we can observe how the extractor behaves on data that it has not observed while it is being trained, thus giving an idea of its generalization capability. Using this split, the classifier is trained on a set of 9537 videos, and evaluated on a set of 3783. It is possible that the division we have used is not the most appropriate, since the validation set is relatively large. In our case, we do not need a validation set that guarantees a fair comparison between our model and the rest of the models that solve this problem, but we want to get an idea of the correct performance of our extractor. It is possible that with a larger training set, in exchange for a smaller validation set, we would achieve higher quality features, since we would have more training diversity. However, we have not studied this detail in more depth because the default division already leaves a set of training data of acceptable size, sufficient for the experimentation we face in this work.

As previously mentioned, in order to train the feature extractor, we modified the last part of the network to solve the classification problem on this dataset. Specifically, to the extractor architecture (composed of the Xception modules for image processing plus the LSTM layer for temporal learning), we add two hidden layers, and the output layer, which has 101 neurons (as many as classes in the dataset). Thus, we have the following final structure:

• Xception module, which accepts images of size 299 × 299 × 3, and returns an image descriptor of size 2048. Since we work with if we have 16 frames, using the TimeDistributed module of Keras we can have 16 copies of the network in parallel, so that the output is a time series of 16 time instants and 2048 features per instant. This part of the network is trained on ImageNet and is not retrained.

• LSTM recurrent layer accepts as input a time series of 16 time instants and 2048 features, and returns a series description of size 512, 768 or 1024, depending on the size of the descriptor we want to extract. This part of the network is randomly initialized and trained at this stage.

• Dense layer classifier. This part of the network, consisting of three layers, processes the temporal descriptor and performs the final classification. Like the recurrent part, it is randomly initialized and entered at this stage. This entire module will be discarded after training, thus retaining the feature extractor part exclusively. The size of the two hidden layers will depend on the size of the descriptor (for example, in the case of descriptor size 1024, the dense layers are of size 512 and 128). The last layer always has 101 neurons, as many as we have classes in the dataset.

To construct the dataset examples, because the original videos have different sizes and durations, the frames have been scaled to have size 299×299 (in our experimentation distortion has not been taken into account in the same way as it is in C3D), and the videos have been sampled to obtain fragments of 16 frames equispaced between the beginning and the end of each video. We therefore obtain a representation for each sequence of 16×299×299×299×3. For the preprocessing of each frame, Keras provides a function that normalizes the input to suit how the Xception model was trained. This function, roughly speaking, performs a normalization of each pixel value by dividing by the standard deviation and subtracting the mean of the pixel values of the ImageNet set.

The Adam optimizer is used for training the model with a learning rate of 10−5, and a decay of 10−6 (this optimizer reduces the learning rate after a certain number of epochs, in order to fine tune the

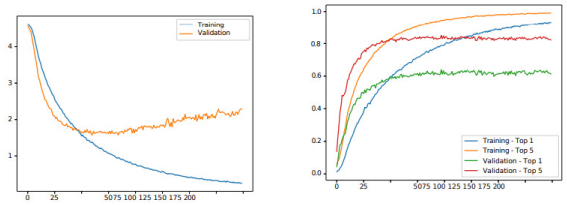
result in the last stages of training). The cost function to be optimized is the categorical entropy, and the model is trained for 200 epochs. To keep the best model found so far, we use the ModelCheckpointer feature of Keras, which allows you to specify a metric and the checking period, and saves a copy of the model at the training time when we check if the metric is the best obtained so far. We use as a metric to observe the accuracy obtained on the validation set. We observe a metric on the validation set so that the overfitting does not lead us to think that the model performs very well, despite a poor generation capacity.

During training, three different metrics have been collected, which are often used in the evaluation of classification problems with such a large number of classes; the value of the cost function, which indicates whether the training is being carried out correctly, the "Top-1" accuracy, which indicates the percentage of correctly classified examples, and the "Top-5" accuracy, which indicates the percentage of examples whose actual class is among the five most probable classes predicted by the model. These last two metrics are particular cases of the "Top-k" accuracy, which is very useful for evaluating problems with a large number of classes. In many cases, the classes represented in data sets of this type have a large overlap because they represent similar concepts. In the predictions of the models, several classes appear with a high probability, very similar for all of them, among which the real class is usually found. The "Top-k" metric partly addresses the fact that the model has selected the correct class as one of the most likely, but not as the most likely of all. Clearly, it is a much less serious error to predict almost equiprobably the classes "Riding a horse" and "Horse racing" (both present in the UCF-101 set), but put slightly above the correct in- than to assign with high probability the class "Playing guitar" high to an example of "Riding a horse". This is why the "Top-k" metric for $k \geqslant 3$ is usually considered to be fairer than the "Top-1" metric for problems with so many classes.

The graphs in Figure 4 show the evolution of these three metrics during the training of the 1024 representation size model. We do not show the evolution of the other two models because the reasoning on them is similar and does not provide new information.

In view of the results that can be observed, we have obtained a fairly good quality feature extractor. We observe how from 100 epochs onwards the model begins to over-fit, since in the cost function graph the value in the training set continues to fall, but in the validation set it stagnates and begins to rise. Furthermore, in the accuracy graph there is a stagnation of the values obtained.

Thanks to the ModelCheckpointer we discussed previously, we have conserved the state of the model at epoch 100 for our feature extractor. We have considered that at that point a quality model is obtained from the metrics we have calculated. At the guard point, we have considered that a quality model is obtained at that point from the metrics we have calculated.



(a) Evolution of the loss function.

(b) Evolution of Top-1 and Top-5 accuracy.

Fig. 4: Proposedfeature extractor, along with fully connected layers for training.

At the previous save point, at 80 epochs, the model performs slightly worse in terms of both Top-1 and Top-5 accuracy, and very similar cost function values. The next save point, at 120 epochs, is already at the stage where the model over-fits, so we are left with the intermediate model.

After training the extractor models, we obtain the following table of accuracies:

Table 1: Top-1 and Top-5 results for the three extractors on UCF-101

Dimensions	Top-1 Accuracy	Top-5 Accuracy
512 elements	57.63 %	82.42 %
768 elements	62.98 %	84.67 %
1024 elements	63.27 %	84.66 %

As we can see, for the three dimensions we obtain a quite good quality. The original C3D model shows in their experiment [2, Figure 2] that

their model reaches a Top-1 accuracy of 45% when trained as a full neural network (as we have done). They then show the results of their final model, which employs the C3D features but uses an SVM for the final classification, and whose results are significantly better, with a Top-1 accuracy of more than 80 %. However, as we are interested in comparing the feature extractors, we believe that the fair comparison should be with the full neural model. From the results obtained, we can conclude that our extractor is more powerful than the one they propose, at least as far as UCF-101 classification is concerned.

Comparing the results of our three extractors, it seems clear that there is an improvement the larger the size of the feature vector. This improvement is not as noticeable between the 768 and the 1024 extractor, whose results are very similar (especially in terms of Top-5 accuracy). However, there is a substantial improvement between these two models and the smaller one.

2) *Feature extraction*

For feature extraction from the UCF-Crime set videos, we followed the same policy as in the original work. We split the video into 16-frame fragments with no overlap, pre-processed the frames to bring them to size 299×299, and used the function provided by Keraspreprocessing for Xception.

Using the network trained according to the previous section without the dense layers, we obtain for each video a representation of size k×1024, with k equal to the number of non-overlapping fragments of 16 frames. By grouping these fragments into 32 groups and taking the average of the fragments in each group, we obtain the final representation of the video consisting of 32 fragments, each with a feature vector of size 1024.

3) *Classifier training*

Once we have the extracted features, we apply a strategy analogous to that of the original model for training the classifier. The classifier is again a fully connected neural network, but smaller in size than the one used by the original model. We have two hidden layers, of size 512 and 64, respectively, and the output neuron. The reduction in the size of the hidden layers is justified by the reduction in the size of the descriptor, which has become half the original size.

In addition, the deactivation rate of the Dropout layers has been reduced from 0.6 to 0.4, as we considered that the original model took an excessive value, which can also cause the classifier performance to degrade and training to be much slower.

The weighting values of the loss function terms have also been adjusted. The value of the kernel regularizer has been increased to 0.01, and the multipliers of the time and sparse constraints have been reduced by half (from 0.00008 to 0.00004). These adjustments have been made after studying the first training runs of the model and observing that it ate too many false negatives. This could occur because the dispersion restrictions were too strong, so we have slightly mitigated that effect by reducing the significance of the term.

The optimizer used in our case is again Adagrad, although we have increased the learning rate to 0.2 at the beginning. It has been observed that the training is quite slow at the beginning, and this modification improves this initial bad behaviour.

C. *Results obtained*

In this section, we show the results obtained in the final experiment. First, we show the confusion matrices for the 0.5 decision threshold of both models:

Table 2: Top-1 and Top-5 results for the three extractors on UCF-101

Model	TN	FP	FN	TP
Original - pre-training	902433	125044	49699	34632
Original - replicated	783342	244135	43699	40632
Xception-LSTM - 1024	875515	151962	44145	40186
Xception-LSTM - 768	908518	118959	52874	31457
Xception-LSTM - 512	914259	113218	60661	23670

In the table we can observe several details that are worth commenting on. First, we can observe how the original model trained by the authors

(marked here as pretrained) and the original model replicated by us do not obtain equivalent results. Our replication has a larger number of true positives, with an increase of 6000.

The results of this experimentation are not very reliable, since it is usually very difficult to replicate the training conditions of a model, and therefore a comparison with the original results may not be entirely fair. This is the reason why we have decided to replicate the experimentation, since it is usually very difficult to replicate the training conditions of a model, and therefore the comparison with the original results may not be entirely fair. The model replicated by us has been trained in conditions similar to those we have imposed on our proposal, so the comparison is, from our point of view, fairer with that model.

Comparing our proposals with each other, we can see how the size of the representation significantly affects the results obtained by the model. The larger the feature vector, the more anomalous frames are detected, at the cost of committing more false positives. However, the increase in false positives is not as noticeable as the increase in well-classified frames, so we believe that our model improves with increasing representation size.

It is especially noteworthy to compare the increase in false positives between models. While for the 512- and 768-characteristic models, the increase in false positives is small (in fact, in absolute terms, the increase in false positives is smaller than the increase in true positives, despite the class imbalance), when we look at the 1024-size model this problem skyrockets. It would be worthwhile to study why this increase has occurred.

As for the comparison between our proposal and the original model, if we compare with our experimentation, the proposed model of larger representation size is significantly better than the original proposal. The number of true positives is almost the same, but the number of false positives has been drastically reduced. This indicates that the features we have extracted are better able to differentiate normal from abnormal fotograms than those extracted by the C3D model. If we compare with the original proposal trained by the original authors, we obtain a considerable increase in the number of true positives (we detect about 6000 more positive frames), but at the cost of a significant increase in false positives. In this case, probably, the decision between choosing one model or the other if we look exclusively at the confusion matrices would be justified by the requirements of the real problem to be solved. In this case, detecting a greater number of positives is advisable, since we are talking about an alarm system.

A false positive will be less serious than committing a false negative, since in the case of a false positive it simply warns of something that is not really happening, but in the case of a false positive we can ignore a dangerous situation that we would be interested in detecting.

In addition to the confusion matrices, the ROC curves and PR curves are shown below:

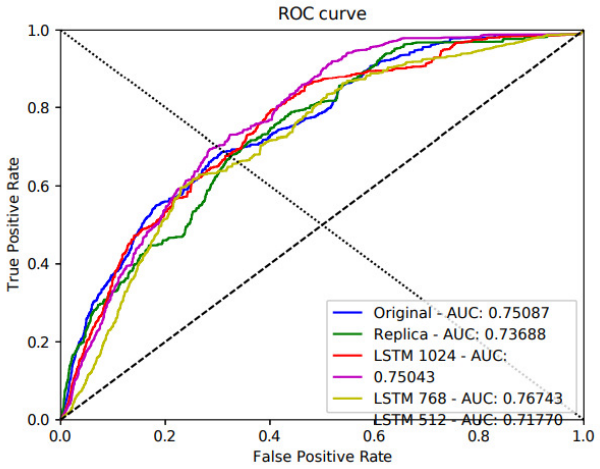


Fig. 5: Proposedfeature extractor, along with fully connected layers for training.

In view of the ROC curves, we can conclude that all models perform similarly. In principle, the two models with the worst ROC curve quality are the replica of the original model and the Xception-LSTM model with 512 feature size. These two curves are dominated by the other 3 practically throughout the graph. On the other hand, the curve obtained by the Xception-LSTM model of size 768 is clearly dominant in most of the graph. For low false positive rates this advantage is not so noticeable, but it quickly becomes the best model and dominates the rest of the curves in the rest of the graph, reached only by the 1024 features representation for a false positive rate around 0.4.

Another detail we can observe is that the models based exclusively on convolution perform better than our models when it comes to low false positive rates. Both the original model and the replica remain the best models when we are talking about a false positive rate below 0.1. However, beyond that point, our proposals achieve similar performance to the original proposals, and quickly outperform them for the rest of the graph.

If we look in terms of AUC, surprisingly, the best model found is the one using the 768 size representation. Although in the confusion matrix we found the 1024 size model to be better, in terms of AUC we have a better performance by the medium size model. In particular, it outperforms by more than 1.5 percentage points both the original model and our proposal for the larger size, both of which achieve very similar results. We are talking about a relatively large improvement, so this model could be the most appropriate in some contexts. In addition, the EER, which is the point where the ROC curve intersects with the diagonal dashed line, also shows that the best model is the LSTM of size 768. Next, with a very similar result between them, are the two full convolutional and the LSTM model of size 1024. Finally, the worst performing model in this case is again the LSTM of size 512.

We move on to show the PR curves:

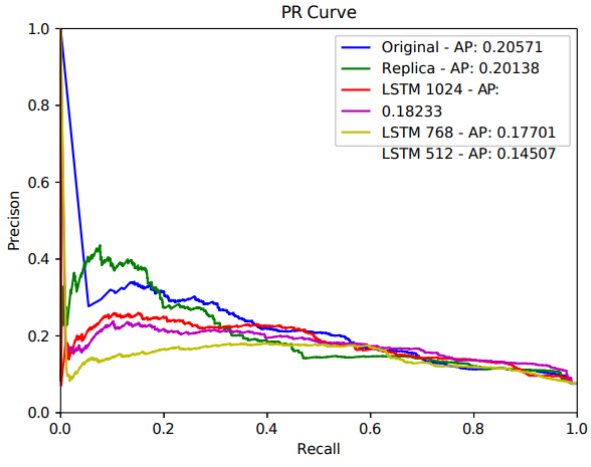


Fig. 6: PR curves of the models

In this comparison, we found very different results. On the one hand, if we look at the left side of the graph, we see that the replica of the original model has the highest accuracy, but here we are talking about low true positive rates (this metric is called Recall, and is the name shown here). We also have the original model above in this case. As we move up the graph towards higher levels of recall, the replica starts to lose much of its credibility, and quickly becomes the worst of the models. Something similar happens to the original model, which ends up being surpassed by the 1024 and 768 models. However, the loss of performance in this case is much more gradual than in the replica.

Our three models, on the other hand, have a similar behaviour throughout the graph. Although they do not reach very high levels of accuracy at any point, the accuracy remains more or less constant. They even outperform the original models in a large part of the graph. However, due to the good performance of the convolutional model at low true positive rates, when we look at the average accuracy both the original and the replica outperform all of our proposals.

This good behaviour at the beginning means that the convolutional models manage to differentiate more clearly the clearly anomalous behaviours than the Xception-LSTM models. That is, for those examples in which the anomaly is clear, the convolutional model tends to respond more reliably, so that it correctly classifies a significant percentage of such frames. On the other hand, when more difficult classification frames begin to appear, the Xception-LSTM models start to perform better than the pure convolutional models.

We will now show and analyse the comparative metrics table:

Table 3: Table of comparative metrics of the models at the photo level

Model	Accuracy	AUC	F1	EER	AP
Original - pre-training	0.8428	0.7508	0.2838	0.3119	0.2057
Original - replicated	0.7411	0.7369	0.2201	0.3253	0.2014
Xception -LSTM - 1024	0.8236	0.7504	0.2907	0.3221	0.1823
Xception -LSTM - 768	0.8455	0.7674	0.2681	0.2980	0.1770
Xception -LSTM - 512	0.8436	0.7177	0.2140	0.3388	0.1451

The first metric we encounter is accuracy. Although we know that this metric is not very relevant when we are faced with a very unbalanced classification problem, such as ours, it is a metric that is usually calculated, so we have decided to include it. For this metric, we have the best performance for the LSTM model of size 768, although almost all models obtain similar results. The original replicated model is clearly the worst of the five in this metric, scoring about 10 percentage points lower than the rest.

In terms of AUC, as we had already seen in the previous graph, the best model is the proposal of size 768, which far outperforms the rest of the models. In particular, it obtains better results than the original model itself in the article. It is important to note that our extractor was pretrained on a set containing less information than the one used in the original extractor, so this margin of improvement that we have obtained could be further enlarged with the new data. As for the rest of the models, we can see how the original model achieves a similar result to that obtained by the 1024 size proposal. Slightly behind is the replica model, and finally the Xception-LSTM model of size 512.

The improvement that we consider most important from our study is on the F metric1 for the 0.5 decision threshold. This improvement was foreseeable given the results we observed on the confusion matrix, but here we confirm that it does indeed occur. In particular, we have improved this metric with the proposed model of size 1024, which achieves the highest score of all models. We consider this metric to be particularly relevant because it represents the behavior of the model.

This implies that, for the set decision threshold, the larger Xception-LSTM model performs better than the original model, which supports our starting hypothesis. This implies that, for the set decision threshold, the larger Xception-LSTM model performs better than the original model, which supports our starting hypothesis. It is curious to note that, despite being the best model in terms of AUC by a significant margin, the model of dimension 768 obtains worse results in this metric. This highlights the importance of using several metrics to compare models, since the use of a single metric, unless we are very interested in a specific behavior, tends to give erroneous ideas.

The last two metrics we have shown can be observed in the graphs shown. The equal error rate (EER), which indicates a better model the lower the value, could be observed in the graph of the ROC curves, since it is the point where the curves crossed the diagonal of points. As we said, the 768 dimension model is the best model in terms of this metric. The AP, which was shown in the PR plot legend, tells us that the best models in terms of average accuracy are the pure convolutional models, because of the very high accuracy they had for low hit rates.

Finally, we show the results in terms of the models ability to classify at the video level. The ability of the models to detect the presence of an anomaly may be even more important than the exact location of the anomaly within the video. Even if the anomaly is not perfectly delimited, i.e., the first or last seconds of the anomaly are not precisely delimited, the fact of generating the alarm when an anomalous situation actually occurs, and not doing so when we are faced with a normal video, is of great importance if we intend to implement a system of these characteristics in a real environment.

The following table shows the results of the models in terms of videolevel evaluation:

Table 4: Percentage of normal and abnormal videos in which an alarm was generated.

Model	% Normal videos	% Anomalous videos
Original	13.33	64.89
Replica	11.11	74.05
Xception-LSTM - 1024	15.55	77.86
Xception-LSTM - 768	12.59	72.52
Xception-LSTM - 512	8.15	71.76

For normal videos, a higher percentage implies that more false alarms have been generated in normal videos, and therefore a worse performance of the model. For anomalous videos, on the other hand, a higher percentage indicates that anomalies have

been detected in a larger number of videos, and therefore better results.

For these metrics, we can observe that our proposal is also of quite good quality. On the one hand, it is observed that the original model has a quite improvable behavior, especially in terms of anomalous unlabeled videos. In particular, no alarms are generated in one out of three videos, which means that it is a rather unreliable model for use in real environments. The original model replicated by us improves these results significantly, generating fewer alarms in normal videos, and a significant increase in alarms for anomalous videos. Specifically, alarms are lost in only one out of four videos. Still a high value, but significantly better than the original.

Comparing the results obtained by our models, we find that the larger the representation size, the more anomaly information is collected. The increase in descriptor size is accompanied by a significant increase in the number of positive videos in which an alarm is generated. As a counterpart, a much higher number of false positives are also committed. The poor performance of the 768-feature model is particularly notable in this case. It barely improves the percentage of detected videos compared to the 512-size model, and on the other hand, it also has almost as many false alarms as the 1024-size model.

Between the 1024 size model and the original model trained by us, which are the two best performers, the decision between which one performs better is subjective. Our model is able to detect a higher number of anomalous videos, but at the cost of a significant increase in the number of false alarms. Probably, if the model is to be used in a real application, we will be more interested in a model similar to ours, since we will be interested in detecting as many anomalous situations as possible. The occurrence of false negatives is less of a concern, since an alarm of this type will probably serve as a warning method and not as a decision making method. We are therefore interested in it being very sensitive to the positive class, and not so much to the negative class, since false alarms will be processed a posteriori by a human.

IV. CONCLUSIONS

We studied the effectiveness of using spatio-temporal features extracted with deep learning models for video anomaly detection. Specifically, we experimented on a crowd anomaly detection model using a feature extractor based exclusively on deep learning models convolutional neural patterns in three dimensions. For such a model, the feature extractor has been replaced by a composite of convolutional and recurrent layers. Our starting hypothesis was that recurrent neural networks would be better temporal feature extractors than 3D convolutional networks.

From the experimental results, we have been able to verify that the combined convolutional-recurrent model performs better than the purely convolutional model for the analysis of video sequences.

Finally, although the results obtained are better than those of the original experimentation, it can be seen that there is still ample room for improvement in this data set. The number of false negatives is still very high, with less than 50 % of the positive photograms being correctly classified. It is possible that this problem is partly justified by the type of labeling of the set. Having to train without the exact location of the anomalies makes it difficult to teach the model to accurately locate the anomaly in the complete anomalous video. This implies that errors are likely being made in the first and last frames around the anomalies. Furthermore, we have seen that in a quarter of the videos labeled as anomalous we did not generate any positive labels, i.e., we ignored almost 25% of the anomalies present in the set. We are talking about a very significant number of errors, which will require more powerful models to be detected.

ACKNOWLEDGMENT

I would like to express my deepest gratitude to a multitude of individuals and organizations whose contributions have been indispensable to the completion of this thesis. The journey of my research, filled with challenges and achievements, was made possible by their generous support and cooperation.

Foremost, I extend my sincerest appreciation to my supervisor, Qing Tian unwavering guidance, encouragement, and expertise. His mentorship has been a beacon of light throughout this journey,

providing me with the academic rigor and moral support needed to navigate the complexities of my research. Qing Tian's patience, knowledge, and commitment have been the foundation of my scholarly growth and achievements.

My gratitude extends to the various institutions, companies, and organizations that have facilitated my research. Special thanks to Nanjing University of Information Science & Technology, and all others for providing the essential tools, resources, and environments crucial for my study. The assistance received has been pivotal in overcoming challenges and achieving my research objectives.

Furthermore, I appreciate all those who provided logistical support, guidance, and encouragement throughout this journey. The collaborative spirit and shared wisdom of these individuals and groups have been a source of motivation and inspiration.

In conclusion, my heartfelt thanks go out to everyone who has played a part in this academic endeavor. Your varied forms of support have not only facilitated my research but have also been vital to my growth as a scholar. Thank you for your invaluable contributions and for being a part of my journey.

REFERENCES

[1] WaqasSultani, Chen Chen y Mubarak Shah. “Real-world anomaly detection in surveillance videos”. En: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pages. 6479-6488.

[2] Du Tran y col. “Learning spatiotemporal features with 3D convolutional networks”. En: Proceedings of the IEEE international conference on computer vision. 2015, pages. 4489-4497.

[3] Andrej Karpathy y col. “Large-scale video classification with convolutional neural networks”. En: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014, pages. 1725-1732.

[4] Sepp Hochreiter y Jürgen Schmidhuber. “Long short-term memory”. En: Neural computation 9.8 (1997), pages. 1735-1780.

[5] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. En: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, pages. 1251-1258.

[6] Jia Deng y col. “Imagenet: A large-scale hierarchical image database”. En: 2009 IEEE conference on computer vision and pattern recognition. Ieee. 2009, pages. 248-255.

[7] François Chollet y col. Keras. https://keras.io. 2015.

[8] Khurram Soomro, Amir Roshan Zamir y Mubarak Shah. “UCF101: A dataset of 101 human actions classes from videos in the wild”. En: arXiv preprint arXiv:1212.0402 (2012).

[9] Gary Bradski y Adrian Kaehler. “OpenCV”. En: Dr. Dobb’s journal of software tools 3 (2000).

[10] Travis E Oliphant. A guide to NumPy. Vol. 1. Trelgol Publishing USA, 2006.

[11] Stefan Van Der Walt, S Chris Colbert y Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. En: Computing in Science & Engineering 13.2 (2011), pages. 22.

[12] The pandas development team. pandas-dev/pandas: Pandas. Ver. latest. Feb. de 2020. doi: 10.5281/zenodo.3509134. url: https://doi.org/10.5281/zenodo.3509134.

