Multimedia Systems (2020) 26:585–605
https://doi.org/10.1007/s00530-020-00667-4

REGULAR PAPER

A real-time two-input stream multi-column multi-stage convolution neural network (TIS-MCMS-CNN) for efficient crowd congestion-level analysis

Santosh Kumar Tripathy[1] · Rajeev Srivastava[1]

Received: 2 January 2020 / Accepted: 20 June 2020 / Published online: 8 July 2020

Abstract
Crowd congestion-level analysis (CCA) is one of the most important tasks of crowd analysis and helps to control crowd disasters. The existing state-of-the-art approaches either utilize spatial features or spatial–temporal texture features to implement the CCA. The state-of-the-art deep-learning approaches utilize a single column convolution neural network (CNN) to extract deep spatial features to solve the objective function and perform better than traditional approaches. But still, the performance is needed to be improved as these models can not capture features invariant to perspective change. The proposed work is mainly based on two intuitions. First, both deep spatial and temporal features are required to improve the performance of the model. Second, a multi-column CNN with different kernel size is capable of capturing features invariant to perspective and scene change. Based on these intuitions, we proposed a two-input stream multi-column multi-stage CNN with parallel end to end training to solve the CCA. Each stream extracts spatial and temporal features from the scene, followed by a fusion layer to enhance the discrimination power of the model. We demonstrated experiments by using publicly available datasets such as PETS-2009, UCSD, UMN. We manually annotated 22 K frames into one of five crowd congestion levels such as Very Low, Low, Medium, High, and Very High. The proposed model achieves accuracies of 96.97%, 97.21%, 98.52%, 98.55%, 97.01% on PETS-2009, UCSD-Ped1, UCSD-Ped2, UMN-Plaza1 and UMN-Plaza2, respectively. The model processes nearly 30 test frames per second and hence applicable in real-time applications. The proposed model outperforms some of the existing state-of-the-art techniques.

Keywords Crowd analysis · Crowd congestion-level classification · Multi-stage multi-column CNN · Feature fusion

1 Introduction

With the massive growth in the worldwide population, efficient crowd management is highly required for public security, safety and also to control crowd disaster. Analysing and understanding crowd is the initial step for effective crowd management. The crowd analysis related tasks include but not limited to crowd count and density estimation [1], crowd behaviour like abnormality detection [2], crowd type detection [3], group activity detection [4], crowd video understanding [5]. Among these, crowd analysis using crowd count and density estimation attracts many researchers in recent years. It becomes the backbone for the crowd analysis related tasks such as crowd abnormal behaviour detection (panic crowd, gathering, running, fighting, over-crowd, and so forth) [6, 7], crowd congestion-level analysis [8], dominant crowd motion direction detection, and many more. Fig. 1 shows a basic workflow diagram for different crowd analysis tasks using crowd count and density estimation. The focus of the proposed work is to provide an efficient model for crowd congestion-level analysis (CCA). The CCA provides a degree of congestions information in a crowd scene, which helps for crowd disaster management. We can implement the CCA at the global-level (Frame-level) or local-level (Patch-Level).

Communicated by C. Xu.

✉ Santosh Kumar Tripathy
santoshktripathy.rs.cse18@iitbhu.ac.in

Rajeev Srivastava
rajeev.cse@iitbhu.ac.in

[1] Computing and Vision Lab, Department of Computer Science and Engineering, Indian Institute of Technology (BHU), Varanasi, India

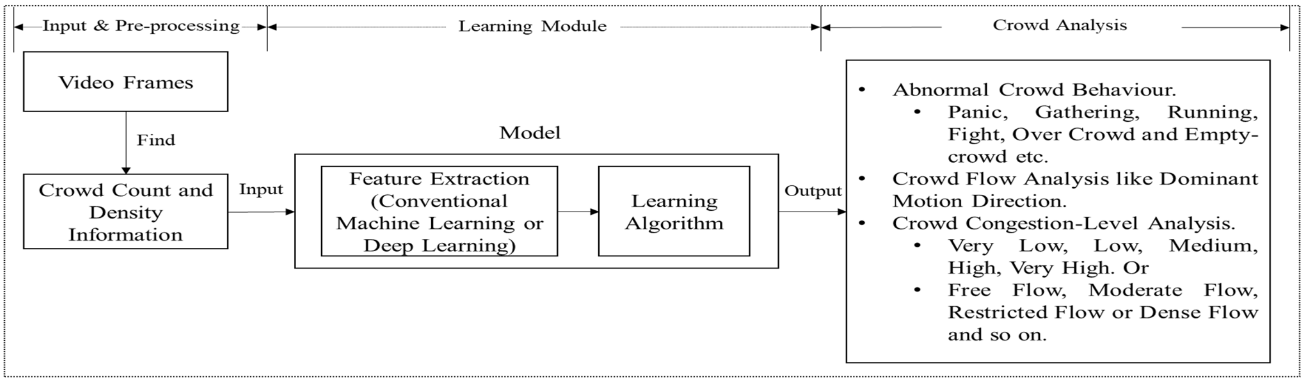
Input & Pre-processing
Learning Module
Crowd Analysis
Video Frames
Find
Crowd Count and Density Information
Input
Model
Feature Extraction (Conventional Machine Learning or Deep Learning)
Learning Algorithm
Output
Abnormal Crowd Behaviour.
Panic, Gathering, Running, Fight, Over Crowd and Empty-crowd etc.
Crowd Flow Analysis like Dominant Motion Direction.
Crowd Congestion-Level Analysis.
Very Low, Low, Medium, High, Very High. Or
Free Flow, Moderate Flow, Restricted Flow or Dense Flow and so on.

Fig. 1 Crowd analysis using crowd count and density estimation

In global-level CCA, the crowd scenes are annotated with several congestion levels/classes with the help of crowd density or crowd flow information followed by feature extraction and classification. But in local-level CCA, the crowd scene is divided into different blocks/patches, and then, these patches are annotated with different congestion classes. The feature extraction and classification are done at the patch-level only. The division of congestion classes is mainly based on service level information provided by Polus et al. [9] or manually defining classes based on density and crowd flow information [10]. For example, the congestion classes could be free-flow, restricted-flow, jammed-flow and dense-flow [9] or very low (VL), low (L), medium (M), high (H) and very high (VH) [10]. Both conventional machine learning and deep-learning approaches have been proposed to solve the objective function. The existing conventional approaches extract spatial (shape, spectral, texture) [11–19] or spatial–temporal texture features [20, 21] to solve the objective function as a multi-class classification problem. But these methods lack in extracting fine-grained features, thereby results in a high misclassification rate and computationally very expensive. The existing deep-learning frameworks utilize CNN architecture [9, 22] to extracts spatial features to solve the objective function. The proposed work is based on two intuitions such as (a) extracting only spatial features won't increase the accuracy since the crowd scene is affected by cluttered background, lighting change, varying crowd densities, perspective change. Moreover, it does not provide any information related to crowd motion. So, in addition to the spatial features, temporal or motion features should be extracted and fused with it. And (b) the single column CNN [9, 22] for CCA cannot capture features invariant to perspective change or scene change, but multi-column CNN [23] with different kernel size is capable of extracting invariant features. Hence based on these two intuitions, we proposed a two-input stream multi-column multi-stage CNN (TIS-MCMS-CNN) to solve CCA in real-time. The two streams extract spatial and temporal features from the crowd scene. These features are fused using a fusion layer, which is followed by two dense connection layers and a classification layer. We perform end to end training, and experiments are done by using publicly available datasets such as PETS-2009 [24], UCSD (Ped1 and Ped2) [25], UMN (Plaza1 and Plaza2) [26]. We divided the dataset into five density classes such as Very Low (VL), Low(L), Medium (M), High (H), Very High (VH) according to work done by Fu et al. [10]. Each density class represents particular congestion information. The experiment results show the robustness of the model and outperform the existing state-of-the-art techniques in terms of accuracy. The proposed model processes nearly 30 test frames per second and shows that it can be appropriate for real-time applications. The idea for TIS-MCMS-CNN is adopted from the single input stream multi-column convolutional neural network (MCNN) [23], which was originally implemented for crowd count using crowd density map estimation.

The remaining part is organized as Sect. 2 describes a brief literature review of state-of-the-art techniques for CCA, Sect. 3 describes proposed work, Sect. 4 explains details of experiments and results, and Sect. 5 describes the conclusion and future work.

2 Literature review

In the literature, crowd density classification and crowd congestion-level analysis (CCA) are interchangeably used. Based on the feature extraction strategy, traditional methods for crowd density classifications mainly classified into two categories, namely spatial approaches and spatial–temporal approaches. Table 1 shows a brief literature review of state-of-the-art techniques for CCA. Generally, the crowd density features such as shape, texture, edge, moments, spectral (Fourier), wavelet features vary significantly between different congestion levels of crowd. This fact attracts researchers to extract significant spatial features to represent crowd

Table 1 A brief comparison of state-of-the-art techniques for the CCA

References	Congestion Classification at		Dataset	No. of Density Levels	Feature Type		Feature Technique	Learning Algorithm	Results (Accuracy)	Advantages	Limitations
	Local-Level	Global-Level			Spatial	Temporal					
[11]	×	√	Own	Five	√	×	GLDM	SOM	76.51%	Accurately recognizes very low-density frames	It couldn't capture variations between different density levels; therefore, the poor performance achieved
[12]	×	√	Own	Five	√	×	GLDM, MFD, Spectral (Fourier)	SOM, Bayesian Classifier, Curve Fitting	GLDM = 85%, MFD = 75% and Spectral features = 70%	They are first to implement MFD for this problem	Can't extract discriminant features, so it results in poor performance
[13]	√	×	Own	Five	√	×	GLDM, MFD, Chebyshev Moments	SOM	Highest classification is 92% for 25 samples using Chebyshev moment	Achieves better results using Chebyshev moments	Small samples for testing. Performance is affected by noise, shadow, and cluttered background
[14]	×	√	Own	Five	√	×	Texture Histogram	SOM	77.32%	Real-time implementation	Poor performance
[15]	×	√	Own	Five	√	×	Histogram statistical features	SVM	87.95%	It handles challenging tasks such as varying viewing angle and lighting changes	It is assumed that all people are of the same size, and they are located in the same horizontal plane, which is not applicable to real-world crowd scenarios
[16]	√	×	Own	Five	√	×	Gradient Orientation Co-Occurrence Matrix	Bag of words using k-means clustering.	96.01%	The model handles scene changes and noisy background	The test cases contain only 2000 patches
[17]	√	×	PETS-2009	Four	√	×	LBP-Co-occurrence matrix	SVM	94.25%	Good crowd descriptor and better performance	Computationally inefficient

Table 1 (continued)

References	Congestion Classification at		Dataset	No. of Density Levels	Feature Type		Feature Technique	Learning Algorithm	Results (Accuracy)	Advantages	Limitations
	Local-Level	Global-Level			Spatial	Temporal					
[20]	×	√	PETS-2009, Gangnam Railway Station	Five	√	√	Normalized Moving area and normalized contrast	MLP with BP	For Pets2009 and Subway accuracies are 80.4%, 93.18%, respectively		The feature descriptor dimension is only two, which is not suitable for large scale datasets
[18]	√	×	PETS-2009	Five	√	×	Subspace LPB features using LDA and PCA	SVM with RBF kernel	89%	Computationally efficient	Poor performance
[21]	×	√	PETS-2009	Four	√	√	SST-LBP	SVM	Around 87%	Good descriptor	Poor performance. Didn't consider static crowd information
[19]	√	×	PETS-2009, UCSD	Four	√	×	CLBP	SVM	92%	Good descriptor	Not applicable to real-time application
[10]	√	×	PETS-2009, Subway	Five	√	×	CNN	Deep Learning	96.03%	Faster implementation	Didn't consider motion information
[22]	√	×	Own	Four	√	×	GoogleNet and ResNet	Deep Learning	84%	Good descriptor	Performance need to be improved

congestion levels for classification. Based on this intuition, Marana et al. [11] argued that texture information of the crowd scene changes very significantly as the crowd density increases from very sparse to very dense. The sparse crowd possesses coarse texture, whereas the dense crowd contains a fine texture. Based on this intuition, Marana et al. [11] extracted texture features using Grey Level Dependency Matrix (GLDM) for five different crowd density levels and classified using the Self Organization Map (SOM) learning algorithm, but results in poor performance. Again, Marana et al. [12] proposed a new technique in which Minkowski Fractal Dimension (MFD) was used to characterize crowd densities and resulted in 75% correct classification using a SOM learning algorithm. Rahmalan et al. [13] extracted texture, shape, and moment features using GLDM, MFD, Translational Invariant Orthonormal Chebyshev Moments (TIOCM), respectively, for different crowd density classes and classified using SOM. But the performance is degraded due to cluttered background, shadow and noise. Marana et al. [14] proposed a real-time crowd density classification method based on texture histogram and low-pass filter for classification correction in a distributed environment but obtained 73.89% accuracy only. Su et al. [15] adopted the Maximally Stable Externally Region (MSER) [27] detector to extract crowd regions and then applied 3D to 2D projection on it, followed by shadow removal. Histogram statistical features were extracted and trained the model using Support Vector Machine (SVM) to classify four crowd congestion levels but results in moderate performance. Ma et al. [16] proposed a patch-level crowd density classification and global-level crowd density estimation technique by using local texture features called Gradient Orientation Co-occurrence Matrix (GOCM). Bag of visual words is created by normalizing GOCM descriptors followed by k-means clustering. The proposed method is capable of handling scene change and background noise. Wang et al. [17] proposed a local density classification approach using texture features using Local Binary Pattern Co-Occurrence Matrix (LBPCM) for grey image and gradient of the image. The model is trained using SVM and achieves 94.25% accuracy but takes more frame processing time and hence not suitable for real-time application.

Kim et al. [20] extracted normalized moving area and normalized contrast to classify crowd density. The main idea was to find a feature descriptor for the moving crowd and the static crowd. The moving area is obtained by employing Combined Local–Global (CLG) optical flow and accumulated magnitude map, and thresholding value is used to find the normalized moving area. The GLCM is used to capture contrast information. Although the idea is noble, it suffers from a high misclassification rate for a varying scene change and lighting change dataset. Yang et al. [21] extracted spatial–temporal features from the moving crowd scene using a sparse-spatial–temporal local binary pattern (SST-LBP) descriptor followed by spectral analysis. Then SVM was used to classify four crowd density levels. Fradi et al. [18] proposed a patch-level crowd density classification technique by extracting LBP features followed by a discriminant subspace analysis using Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA). The feature descriptors were fed to a modified algorithm for multi-class classification using SVM with RBF kernel. Lamba et al. [28] proposed a technique to extract rotational invariant spatial–temporal LBP features for the moving crowd. At first, key interest points were detected by using Hessian Detector [29] for a volume of frames. Then RIST-LBP features were extracted from the volume of spatial–temporal key points, and then, SVM was employed to classify four crowd density levels. The authors didn't consider descriptors for the static crowd in the scene. Recently, Alanazi et al. [19] proposed a local crowd density classification technique in which CLBP features were utilized to describe four different density levels. Multi-class SVM was used for classification, although the method showed some promising results but can't applicable in real-time as single-frame processing time is 7 s.

A few methods have been proposed using a deep-learning algorithm for CCA. Fu et al. [10] proposed three deep CNN models, namely, modified multi-stage CNN (MS-CNN), optimized CNN, and cascaded CNN. The main focus was to extract in-depth spatial features for CCA and increase the processing speed by deleting wights of the neurons which have similar receptive fields. But the proposed model has some limitations like (a) We have to manually identify hard samples from the dataset for cascade CNN, which is quite impossible to find in real-world applications and (b) Extracting only spatial features can't increase the performance as the crowd scene contains both spatial (for the static crowd) and temporal (moving crowd) information. Pu et al. [22] utilized transfer learning mechanisms to solve the objective. GoogleNet [30] and VGGNet-16 [31] were adopted to classify crowd densities for three and five classes. But still, performance needs to be improved.

In contrast to CCA, some constructive works have done in the field of social image understanding using deep learning. For example, Li et al. [32] proposed a new distance metric called a weakly supervised deep metric learning (WDML) algorithm that discovers the knowledge between the social image visual content and the user-provided tags. Again, Li et al. [33] proposed a deep collaborative embedding (DCE) model to uncover the hidden space for images and their associated tags. As of now, the discussion is limited to the CCA, but it can be extended to Image Understanding related tasks like content-based image retrieval, tag-based image retrieval in our future study.

Now, by observing the literature review on the CCA, we may summarize that,

- Most of the traditional methods either extracts spatial features or spatial–temporal texture features from the crowd scene but lacks in providing better accuracy in real-time.
- The spatial features solely cannot increase the accuracy of the CCA; crowd motion information must be used.
- The deep models provide better accuracy as compared to traditional methods.
- The existing deep approaches only extract in-depth spatial features to solve the objective in real-time but fail to extract invariant features for perspective change or scene change.

The above reasons motivate us to develop a deep model that extracts and fuses invariant deep spatial as well as temporal features for the CCA. Based on this intuition, we proposed a two-input stream multi-column multi-stage CNN (TIS-MCMS-CNN) followed by a fusion layer to perform global-level crowd density classification. The frames are annotated with one of five density classes such as Very Low (VL), Low (L), Medium (M), High (H), and Very High (VH). We follow the work of Fu et al. [10] to annotate these density labels to frames. The inputs to each stream are the original frame and flow magnitude map.

The novelty or main contribution is as follows

(a) Designing a spatial–temporal deep model for the CCA. To be the best of our knowledge, the proposed deep model is the first model for the CCA, which extracts deep spatial–temporal features.
(b) Minutely designed the complex TIS-MCMS-CNN architecture, which processes the frames at the rate of around 30 frames per second.
(c) Extensive experiments were performed to show the robustness of the proposed model.

Section 3 discusses the proposed work in details.

3 Proposed work

To develop an efficient model for the CCA, the model should extract invariant features representing not only global scene information (Spatial) but also moving crowd (Motion/Temporal) information. It is because the crowd scene contains cluttered background, high occlusion, dynamic texture, lighting changes, dynamic crowd shape between frames, static and motion crowds. So, extracting only spatial features wouldn't help to capture more meaningful information to improve accuracy. Hence, crowd motion features should be extracted. Then, fusion of both spatial and temporal (Spatial–Temporal) features should increase the performance. In such cases, conventional machine learning is not capable of extracting discriminant features (concluded from literature review), and hence, deep learning could be the right choice. The existing deep CNN models couldn't provide better accuracies in the presence of perspective change, cluttered background, illumination change. Zhang et al. [23] argued and showed that a multi-column multi-stage CNN with different receptive fields (kernel size) can extract features which are adaptive and invariant to these challenges. Based on this intuition, we proposed a two-input stream multi-column multi-stage convolutional neural network (TIS-MCMS-CNN), which not only extracts deep spatial and temporal features but also fuses them to solve the objective function. Figs. 2 and 3 show the overall architecture and the detail architecture of the TIS-MCMS-CNN, respectively.

The following subsections explain details of the proposed model and its working principle:

- Network architecture.
- Pre-processing and motion magnitude map extraction.
- Problem formulation and learning algorithm.
- Precaution to handle overfitting.

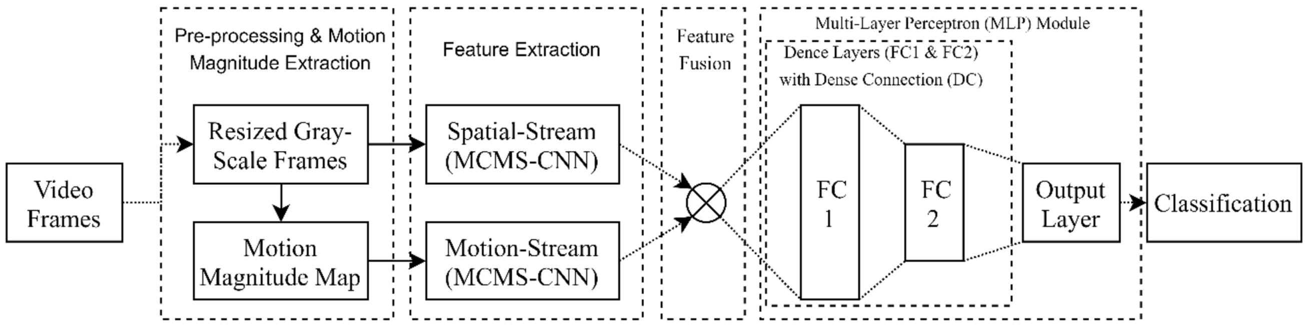
Pre-processing & Motion Magnitude Extraction
Feature Extraction
Feature Fusion
Multi-Layer Perceptron (MLP) Module
Dence Layers (FC1 & FC2) with Dense Connection (DC)
Video Frames
Resized Gray-Scale Frames
Motion Magnitude Map
Spatial-Stream (MCMS-CNN)
Motion-Stream (MCMS-CNN)
FC 1
FC 2
Output Layer
Classification

Fig. 2 Overall architecture of the proposed model

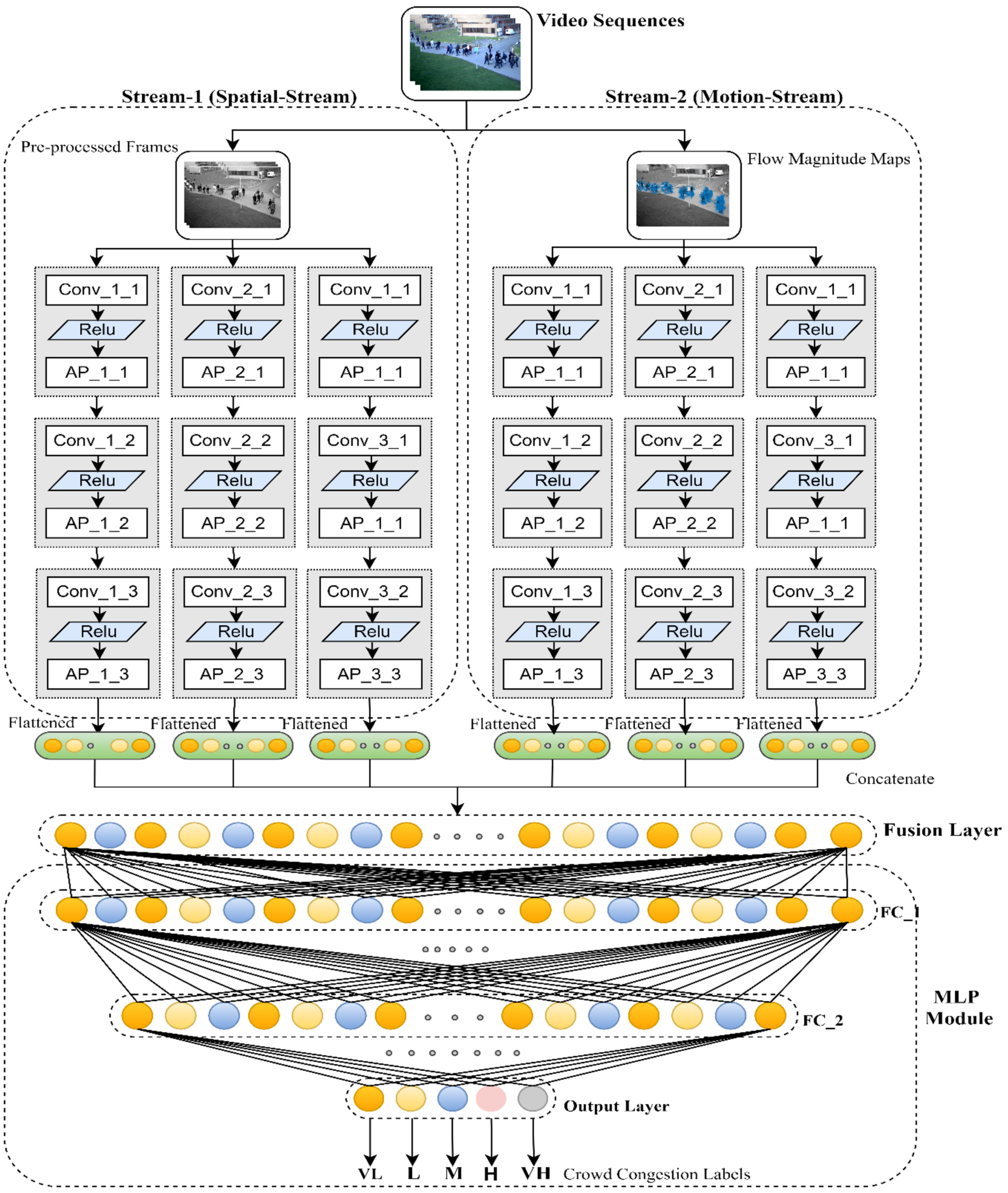

Fig. 3 Detail architecture of the model TIS-MCMS-CNN

3.1 Network Architecture

According to Fig. 3, the proposed architecture contains three main modules such as

- Two streams of multi-column multi-stage CNN.
- A fusion layer.
- A multi-layer perceptron (MLP) module.

The two streams (see Fig. 3) named as stream-1 (the spatial-stream) and stream-2 (the motion-stream). Each stream contains three columns of convolution layers. Each column contains three stages of convolution layers of different receptive field or kernel size. The details of the layer information are given in Table 2. The activation function for each convolution (Conv) layer is a rectified linear unit (ReLU), which is followed by a max pooling (MP) layer. We have taken ReLU for Conv layers, because it performs better than logistic, tanh activation functions, and get control of the vanishing gradient problem. The ReLU for *kth* neuron at level *l* can be calculated using the following Eq. 1.

$$\mathrm{ReLU}\left(\left(\left(z_{\mathrm{in}_k}\right)\right)\right)_l = \max\left\{0, z_{\mathrm{in}_k}\right\}, \tag{1}$$

where $\left(z_{\mathrm{in}_k}\right)$ is the weighted sum of the information transmitted from neurons of level *l-1* to the *kth* neuron of level *l*.

A fusion layer follows the two streams. The feature maps obtained from the third stage of each column are flattened and concatenated in the fusion layer.

Next, an MLP module follows the fused layer. The MLP module contains three dense connection layers, namely fully connection layer-1 (FC-1), fully connection layer-2 (FC-2), and an output layer. The activation function for FC-1 and FC-2 are ReLU. The output layer containing five neurons; each of these neurons is responsible for giving one of five responses such as VL, L, M, H, VH. The activation function for fully connected layers except the output layer is ReLU. The activation of the output layer is SoftMax. The SoftMax function generally gives the probability distribution of each target class. So, we have used this function at the output layer. The following Eq. 2 shows the SoftMax activation for the *pth* neuron of the seventh layer (output) of our proposed model.

$$y_{7_{pout}} = \mathrm{SoftMax}\left(\left(z_{\mathrm{in}_p}\right)_7\right) = \frac{e^{\left(z_{\mathrm{in}_p}\right)_7}}{\sum_{p=1}^{5} e^{\left(z_{\mathrm{in}_p}\right)_7}}, \text{ for } p = 1,2,3,4,5, \tag{2}$$

where $\left(z_{\mathrm{in}_p}\right)_7$ is the weighted information transmitted from the sixth layer to the *p*th neuron of the output layer (seventh layer).

3.2 Pre-processing and motion magnitude map extraction

In the pre-processing stage, we converted each colour frame into its Grayscale and resized to 42×40. Let, the RGB video frames and it's resized grayscale images are denoted using set $VF = \left\{vf_1, vf_2, \ldots, vf_T\right\}$ and $GF = \left\{gf_1, gf_2, \ldots, gf_T\right\}$ respectively, where T is the total number of frames. Eq. 3 is used to convert the colour frames into grayscale images.

$$\begin{aligned} gf_i = {} & 0.299 \times R\left(vf_i\right) + 0.587 \times G\left(vf_i\right) \\ & + 0.114 \times B\left(vf_i\right), \forall i = 1, 2, \ldots, T, \end{aligned} \tag{3}$$

where $R(), G(),$ and $B()$ are Red, Green, and Blue channels of the colour frame, respectively. The motive behind this pre-processing is to minimize the total memory occupancy of the model, and the idea is adopted from Fu et al. [10]. Then, the motion magnitude map of the resized frames is obtained by applying the Lucas–Kanade [34] optical flow. The Lucas–Kanade method cannot give dense-flow information but still provides noise-free motion information. It finds an optical flow between two consecutive frames by solving the following constrained equation.

$$gf_x \times u + gf_y \times v + gf_d = 0, \tag{4}$$

gf_x and gf_y are the spatial derivatives of the *dth* frame, gf_T is the temporal derivative of the *dth* frame, and u, v are the horizontal and vertical optical flow of that frame, respectively. The flow magnitude can be obtained by solving the following Eq. 5.

$$\mathrm{Mag}(x, y, d) = \sqrt{u(x, y)^2 + v(x, y)^2} \tag{5}$$

where $Mag(x, y, d)$ refers to the motion magnitude map for the *dth* frame. The detailed description and solution of Eqs. 4 and 5 can be found in [34]. Let the motion magnitude maps for the set GF is denoted as set $MF = \left\{mf_1, mf_2, \ldots, mf_T\right\}$. The resized grayscale frame set GF and the motion magnitude set MF are inputted to the first stream, i.e. the

Table 2 TIS-MCMS-CNN layers information

Layer name	Filter size	Number of filters	Layer name	Filter size	Number of filters	Layer name	Filter size	Number of filters
Conv_1_1	5×5	25	Conv_2_1	4×4	25	Conv_3_1	5×5	25
Conv_1_2	3×3	15	Conv_2_2	3×3	15	Conv_3_2	4×4	15
Conv_1_3	3×3	10	Conv_2_3	3×3	10	Conv_3_3	2×2	10

Layer name	Filter size	Layer name	Filter size	Layer name	Filter size	Layer name	Neurons
MP_1_1	2×2	MP_2_1	2×2	MP_3_1	2×2	FC-1	512
MP_1_2	2×2	MP_2_2	2×2	MP_3_2	2×2	FC-2	64
MP_1_3	2×2	MP_2_3	2×2	MP_3_3	2×2	Output	5

spatial-stream and second stream, i.e. motion-stream of TIS-MCMS-CNN, respectively. The detailed description of the problem formulation and learning algorithm is discussed in the following subsection.

3.3 Problem formulation and learning algorithm

We trained the network using backpropagation [35] with Adam optimizer [36]. The algorithm-1 shows steps followed during training. For training the network, broadly, we need two things forward propagation and backward propagation.

The network is trained until early-stopping, or itr = max_iteration is satisfied. The early-stopping is a measure used to minimize the overfitting and can be found in Keras as an in-built function. In early-stopping, we need to set patient parameter p, and during training, the model checks whether the validation loss or training loss is/is not going down even after crossing the p number of epochs. If it is going down, then the training continues otherwise stops.

Let *GF* and *MF* are divided into N number of batches. Let tuple $S_{t_x} = \langle \mathrm{GF}_{t_x}, \mathrm{MF}_{t_x} \rangle$ represents pre-processed Gray frames and corresponding motion magnitude maps for the *tth* batch. Here, for any value of *t* (ranges from 1 to *N*) *x* ranges from *1* to Batch_Size. The *Batch*_Size defines the batch size of *tth* batch samples. We assigned resized grayscale frames, i.e. GF_{t_x} and motion magnitude maps of frames, i.e. MF_{t_x} to, the first and second stream, respectively.

During forward propagation, for each sample in S_{t_x}, the feature maps of the two-stream CNNs, the activation of the hidden layers of MLP and the predicted outputs of the final layer are calculated. The forward propagation for the net is discussed below.

The convolution layers of the two-stream CNNs convolves the input matrix with filters and generate feature maps. For the proposed network, the convolution operation of the *ith* layer for the *jth* column of *kth* stream is denoted as,

$$\left[f_{i,k}^{j}\right]^{S_{t_x}} = \left[\mathrm{CONV}\left(\theta_{C_{i,k}}^{j}, \left[fm_{i-1,k}^{j}\right]^{T}\right)\right]^{S_{t_x}}, \tag{6}$$

where θ_C represents parameters for two-stream CNNs and $\theta_{C_{i,k}}^{j} = \left[W_{i1,k}^{j}, W_{i2,k}^{j}, \dots, W_{iM,k}^{j}\right]$ for $i = 1,2,3$ $j = 1,2,3$ and $k = 1,2$. Each $W_{im,k}^{j}$ represents *mth* convolution kernel's weight matrix for *ith* layer of *jth* column of *kth* stream and we got $m = 1,2,\dots,M$ such kernel parameters for each layer. Remember that the value of *M* is different for different layers. The $f_{i,k}^{j}$ is the preactivated feature map. The $fm_{i-1,k}^{j}$ is the activated and max pooled feature map obtained from $(i-1)th$ stage of *jth* column of the *kth* stream. Note that, $fm_{0,1}^{0}$ and $fm_{0,2}^{0}$ represent the inputs to the first and second streams, which are GF_{t_x} and MF_{t_x} respectively. Each stage consists of a convolution layer followed by ReLU, followed by max pooling. The feature map of each stage can be calculated as,

$$\left[fm_{i,k}^{j}\right]^{S_{t_x}} = \left[\mathrm{MP}\left(\mathrm{ReLU}\left(f_{i,k}^{j}\right)\right)\right]^{S_{t_x}}. \tag{7}$$

The feature maps obtained after the third stage of all three columns of two streams are concatenated using a fusion layer, and it can be denoted as

$$[\mathrm{Fuse}]^{S_{t_x}} = \left[\mathrm{CONCATE}\left(fm_{3,k}^{j}\right)\right]^{S_{t_x}}. \tag{8}$$

It should be noted that the fusion layer only concatenates the flattened feature maps of its previous layer; hence there is no weight updating during backpropagation. The next stage of the forward propagation is to find the activation of the neurons of the MLP. The pre-activation for the fully connected layers (5–7 layers) can be calculated as

$$\left[y_i\right]^{S_{t_x}} = \left[\theta_{fc_i} \times \left[y_{(i-1)_{out}}, 1\right]^{T}\right]^{S_{t_x}}, \tag{9}$$

where $\theta_{fc_i} = \left[\omega_i\right]$, ω_i is the weight matrix connecting neurons from $(i-1)th$ layer to the *ith* layer. The y_i and $y_{i_{\mathrm{out}}}$ represent pre-activation and the activated neurons for *ith* layer. Remember that $y_4 = y_{4_{\mathrm{out}}} = \mathrm{Fuse}$. The response of the first two fully connected layers of MLP is ReLU, and the activation can be calculated as

$$\left[y_{i_{\mathrm{out}}}\right]^{S_{t_x}} = \left[\mathrm{ReLU}\left(y_i\right)\right]^{S_{t_x}}, \text{ for } i = 5, 6. \tag{10}$$

The last layer of the MLP is the classification layer, which contains five neurons. The SoftMax activation is used in this layer, and it can be represented as

$$\left[y_{i_{\mathrm{out}}}\right]^{S_{t_x}} = \left[\bigcup_{p=1}^{5}\left[\mathrm{SoftMax}\left(y_{i_{p\mathrm{out}}}\right)\right]\right]^{S_{t_x}}, \text{ for } i = 7. \tag{11}$$

Let $\emptyset_{\mathrm{Net}} = \left[\theta_{\mathrm{C}}, \theta_{\mathrm{fc}}\right]$ represent all the parameters of the network. At last, the loss is computed by using Cross-Entropy between the true distribution and predicted distribution. The cross-entropy loss is a way to find the difference between two distributions like true distribution, i.e. T_{p} and predicted distribution, i.e. $y_{i_{out}}$. So, the loss $L\left(T_{\mathrm{p}}, y_{i_{\mathrm{out}}}\right)$, can be calculated using the following Eq. 12.

$$\left[L\left(\emptyset_{\mathrm{Net}}\right)\right]^{S_{t_x}} = \left[L\left(T_p, y_{i_{\mathrm{out}}}\right)\right]^{S_{t_x}} = \left[-\sum_{p=0}^{4} T_p \log y_{7_{p\mathrm{out}}}\right]^{S_{t_x}}, \tag{12}$$

where $T_p\big|_{p=0,1,2,3,4}$ is the true distribution of the five density classes, namely VL (0), *L* (1), *M* (2), *H* (3), and VH (4) and $y_{7_{\mathrm{out}}}$ is their predicted distribution. Whenever the true distribution and predicted distribution are the same, then the loss function $L(p, q)$ is minimized. So, we want to find predicted distribution such that $-\sum_i p_i \log q_i$ is minimized. So, the problem for the proposed work can be formulated as an optimization problem which minimizes the loss between the true distribution and the predicted distribution, and it can be represented as,

$$\left[\underset{\emptyset_{\text{Net}}}{\arg\min} - \sum_{p=0}^{4} T_p \log y_{7_{p_{\text{out}}}} \mid \sum_{p=1}^{5} y_{i_{p_{\text{out}}}} = 1\right]^{S_{t_x}} \tag{13}$$

Using Lagrangian constraint multiplier, the objective function can be reduced to the following form:

$$\left[\underset{\emptyset_{\text{Net}}}{\arg\min} \sum_{p=0}^{4} -T_p \log y_{7_{p_{\text{out}}}} + \lambda \sum_{p=1}^{5} y_{i_{p_{\text{out}}}} - 1\right]^{S_{t_x}}, \tag{14}$$

where λ is the Lagrangian multiplier. If $\lambda \sum_{p=0}^{4} y_{i_{p_{\text{out}}}} - 1 = 0$, then we will get an absolute minimum.

We have used the L_2 norm as the regularization term and added in the optimization function. So, the loss function could look like as

$$\left[\tilde{L}\left(\emptyset_{\text{Net}}\right)\right]^{S_{t_x}} = \left[L\left(\emptyset_{\text{Net}}\right) + \frac{\alpha}{2}\emptyset_{\text{Net}}^2\right]^{S_{t_x}}, \tag{15}$$

where α is the regularized parameter.

Algorithm-1: Training and optimizing the TIS-MCMS-CNN model

Dataset Preparation: Manually annotate frames into one of five density labels like Very Low (VL), Low (L), Medium (M), High (H) and Very High (VH) (See Table-3) and divide into training and test cases.

Input: Video Sequences or frames.

Output: Optimized TIS-MCMS-CNN.

Steps:

```
1.  Extract frames.
2.  Do pre-processing on frames like RGB to Grayscale conversion using equation-3 and resize to 42×40
    and represented as set GF.
3.  Extract motion magnitude maps from consecutive pre-processed frames by using Lucas-Kanade [34]
    optical flow and denoted as MF.
4.  Divide GF and MF into batches of size $Batch\_Size$. Let there are N number of batches. Let $S_{t_x} =< GF_{t_x}, MF_{t_x} >$ contains pre-processed Gray frames and corresponding motion magnitude maps for $t^{th}$
    batch. Here, t = $1$ $to$ $N$ ,and $x = 1$ $to$ $Batch\_Size$.
5.  Initialize network parameters, hyper-parameters (See Section-4.2) and $max\ \_iteration = 500$.
6.  While Early-Stopping or $itr = \max\ \_iteration$ is satisfied
7.    For each batch t = 1 to N
8.      For each Sample S in batch t i.e. $S_{t_x} where\ x = 1\ to\ Batch\_Size$
9.          Assign resized grayscale frames i.e. $GF_{t_x}$ to the first stream.
10.         Assign motion magnitude maps of frames i.e. $MF_{t_x}$ to the second stream.
11.        // Forward Propagation in Two Stream Multi Column Multi Stage CNN.
12.         Perform convolution operation using equation-6 for each layer of two stream
13.         Find the feature map at every stage of two stream using equation-7.
14.         Fuse the feature maps of two stream using equation-8.
15.        // Forward Propagation in MLP
16.         Perform Pre-activation of layers in MLP using equation-9.
17.         Find activation of layers using equation-10.
18.         Find the activation of output layer using equation-11.
19.         Find the loss by using equation-15.
20.        // Backward Propagation
21.         Find the gradients of loss for the network by solving the equation-16 and 17 using [35].
22.         Find the cumulative gradients by using equations-18 and 19.
23.     End For
24.     Find Cumulative History of gradients by solving equation-20 to 22 using [36].
25.     Update the network parameters by using equation-23[36].
26.   End For
27. END While
28. Save the trained model for testing.
```

During backpropagation, we find out the gradients of the loss using the backpropagation algorithm [35] and update weights and biases using [36]. The gradients with respect to the parameter are calculated at every layer of the proposed model. Let the computed gradients on sample S_{t_x} for the whole network is represented as $[\nabla\emptyset]^{S_{t_x}} = [\nabla\theta_{\mathrm{fc}}, \nabla\theta_{\mathrm{c}}]^{S_{t_x}}$. The gradients with respect to parameters for the MLP layers and two-stream CNNs for a given *Batch*_Size can be calculated by iteratively solving the following Eqs. 16 and 17, respectively.

$$[\nabla\theta_{\mathrm{fc}}]^{S_{t_x}} = [\nabla_{\theta_{\mathrm{fc}}}\tilde{L}(\emptyset_{\mathrm{Net}})]^{S_{t_x}} \tag{16}$$

$$[\nabla\theta_{\mathrm{C}}]^{S_{t_x}} = [\nabla_{\theta_{\mathrm{fc}}}\tilde{L}(\emptyset_{\mathrm{Net}})]^{S_{t_x}} \tag{17}$$

Now, the cumulative gradients $[\nabla\emptyset]^t = [[\nabla\theta_{fc}, \nabla\theta_c]]^t$ for a given batch t can be calculated using the following Eqs. 18 and 19.

$$[\nabla\theta_{fc}]^t = \sum_{x=1}^{\mathrm{Batch\_Size}} \left[\nabla_{\theta_{fc}}\tilde{L}(\emptyset_{\mathrm{Net}})\right]^{S_{t_x}} \tag{18}$$

$$[\nabla\theta_{\mathrm{C}}]^t = \sum_{x=1}^{\mathrm{Batch\_Size}} [\nabla_{\theta_{\mathrm{C}}}\tilde{L}(\emptyset_{\mathrm{Net}})]^{S_{t_x}}. \tag{19}$$

After finding all the gradients, we updated the parameters using Adaptive Moment (Adam) [36] update rule. To solve the decay problem, Adam [36] utilizes the cumulative history of gradients to update the parameters of the network, such as weights and biases. For a given iteration t the cumulative history of gradients can be calculated using following Eqs. (20–22).

$$m^t = \beta_1 \times m^{t-1} + (1-\beta_1) \times [\nabla\emptyset]^t \tag{20}$$

$$v^t = \beta_2 \times v^{t-1} + (1-\beta_2) \times \left([\nabla\emptyset]^t\right)^2 \tag{21}$$

$$\hat{m}^t = \frac{m^t}{1-\beta_1^t} \text{ and } \hat{v}^t = \frac{v^t}{1-\beta_2^t}, \tag{22}$$

where $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Now, the parameters are updated using the following equation:

$$[\nabla\emptyset]_{t+1} = [\nabla\emptyset]_t - \frac{\eta}{\sqrt{\hat{v}^t + \varepsilon}} \times \hat{m}^t, \tag{23}$$

where η is the learning rate.

3.4 Precautions to handle overfitting

Overfitting occurs when the model is overly complex to fit the training-set tightly, which results in remembering the training-set but not learning it. It is because of the complex model always possesses low bias and high variance. So, overfitting is the model's error. If the model is simple, then underfitting may occur. In such a case, the model has high bias and low variance. So, there is always a trade-off between bias and variance. So, why should we care about bias-variance trade-off and model complexity for the deep neural network? Answers to this question are,

- Deep Neural Networks are highly complex models.
- It has many parameters and many non-linearities.
- So, it is easy for them to overfit and drive training error to zero.

Hence, we need some regularization. The followings are different forms of regularization, which help to handle the overfitting.

- L_2 Regularization.
- Early-stopping.
- Drop-out.
- Parameter sharing and tying.
- Data augmentation.
- Ensemble.
- Adding noise to inputs and outputs.

In our proposed model, we implemented L_2 regularization and early-stopping to handle with overfitting. Larger weights in the neural network make the model overfit such that for a small change in the input (during testing), it results in more significant variation in the output. So, we must penalize the weights. For this, L_1 or L_2 regularization can be used. We have used *the* L_2 norm as the regularization term and added in the optimization function.

4 Experiments and results

The proposed model is implemented in PyCharm using Keras and Tensorflow. The model is implemented in the intel-i7 8th generation processor having 8 GB RAM. The experiments are done on three publicly available datasets, namely PETS-2009 [24], UCSD [25] and UMN [26]. To show the efficiency of the proposed model, three state-of-the-art techniques (MS-CNN [10], CLBP [19] and MLP [20]) are implemented and compared. Apart from this we also performed an ablations study in which we implemented each stream, i.e. spatial-stream and motion-stream individually and the results were compared. The following subsections describe dataset preparation, default network parameter values, performance metrics, results analysis and ablation study in detail.

4.1 Dataset Preparation

For the experiment, we have used publicly available datasets, namely PETS-2009 [24], UCSD [25], UMN [26]. The PETS-2009 [24] dataset is the benchmark dataset for crowd surveillance. The PETS-2009 S1 mainly meant for crowd count and density estimation. It contains three scenarios, such as L_1, L_2, and L_3. Each of them contains sequences recorded in four views in two timestamps. We selected all the view-1 of the three scenarios, which were recorded in different lighting and weather conditions. Each view contains a sparse to the dense crowd. The UCSD [25] is a crowd anomaly dataset that provides two sequences in two different scenarios, namely, UCSD-Ped1 and UCSD-Ped2. The Ped1 and Ped2 contain 16,000 and 4800 frames, respectively. These datasets contain different challenging conditions like varying lighting conditions, occlusions, camera jitters. The UMN [26] provides benchmark datasets for crowd monitoring. The UMN-Plaza1 and Plaza2 are two surveillance videos recorded in Plaza1 and Plaza2 for crowd monitoring. These videos are converted into frames, as shown in the table. The frames are manually annotated with one of five density levels according to the degree of congestion of the crowd. The division of crowd scenes (PETS-2009, UCSD) into five density levels such as VL, L, M, H, and VH is motivated by the work done by Fu et al. [10]. The detail of the division of crowd density levels

Table 3 Details of five congestion levels

Datasets	Defining class level densities				
	Very Low (VL)	Low (L)	Medium (M)	High (H)	Very high (VH)
UCSD-Ped1	0–8	9–16	17–24	24–32	>32
UCSD-Ped2	0–8	9–16	17–24	24–32	>32
PETS-2009	0–8	9–16	17–24	24–32	>32
UMN-Plaza1	0	1–3	4–6	7–10	>10
UMN-Plaza2	0	1–3	4–6	7–10	>10

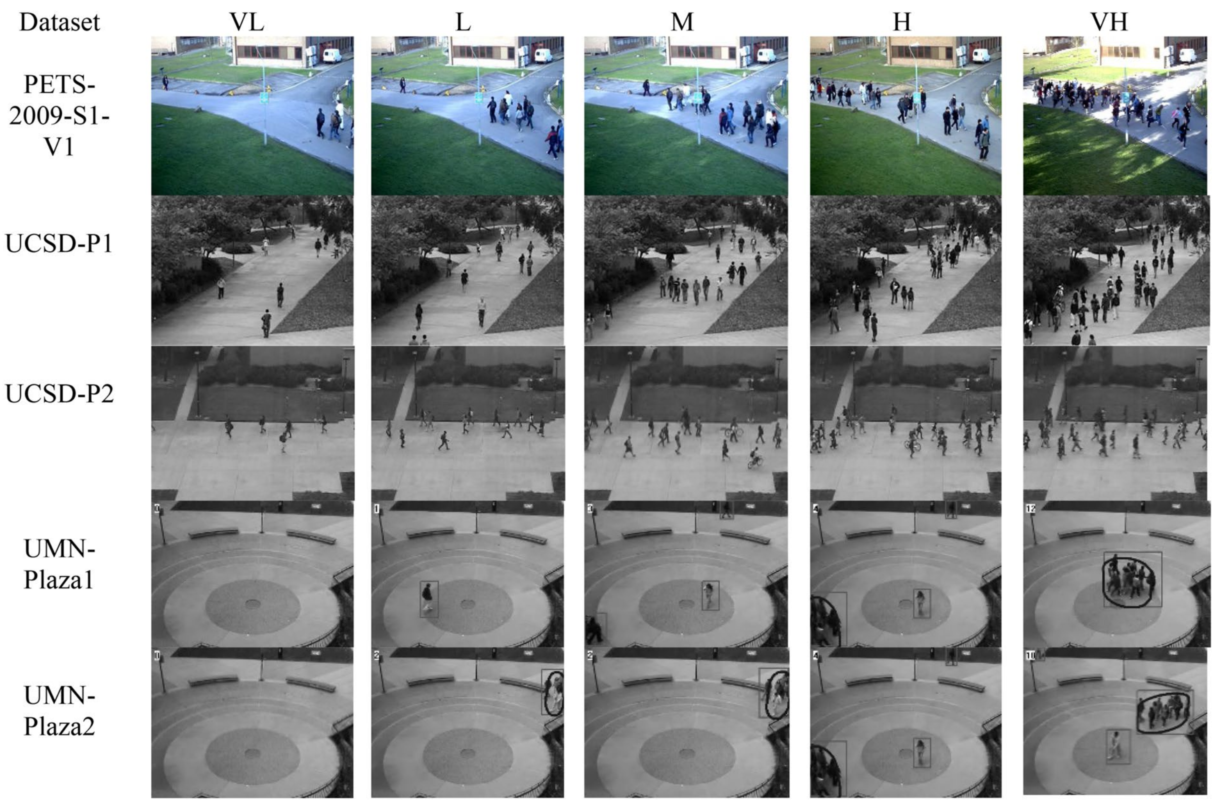

Fig. 4 Examples of crowd scenes of different density levels

Table 4 Details of "Dataset-1"

Dataset Name	Training samples						Testing samples					
	VL	L	M	H	VH	Total	VL	L	M	H	VH	Total
UCSD-Ped1	1713	3808	1462	1006	409	8398	1143	2539	975	671	274	5602
UCSD-Ped2	221	1326	669	322	195	2733	148	885	447	216	131	1827
UMN-Plaza1	116	275	132	65	234	822	78	184	89	44	157	552
UMN_Plaza2	91	373	231	234	71	1000	62	249	154	156	48	669
PETS-2009	52	116	165	102	301	736	35	78	110	68	202	493

Table 5 Details of "Dataset-2"

Dataset name	Training-set					Validation-set					Testing-set				
	VL	L	M	H	VH	VL	L	M	H	VH	VL	L	M	H	VH
UCSD-Ped1	1370	3046	1169	804	327	343	762	293	202	82	1143	2539	975	671	274
UCSD-Ped2	176	1060	535	257	156	45	266	134	65	39	148	885	447	216	131
UMN-Plaza1	92	220	105	52	187	13	55	27	24	47	78	184	89	44	157
UMN-Plaza2	72	298	184	187	56	19	75	47	47	15	62	249	154	156	48
PETS-2009	41	92	132	81	240	11	24	33	21	61	35	78	110	68	202

Table 6 Confusion matrix for binary classification

Actual Vs predicted	Predicted classes		Total number of instances
	Positive	Negative	
Actual class			
Positive	TP	FN	P = TP + FN
Negative	FP	TN	N = FP + TN

TP true positive, *FP* false positive, *FN* false negative, *TN* true negative

is given in the following Table 3. Each entry of Table 3 shows the range of people for that density class. The UMN-Plaza1 and Plaza2 contain the very low crowd, so we divided their density level according to the value shown in the following Table 3. Fig. 4 shows examples of different congestion levels defined on the following datasets.

Generally, the recent works on the CCA did not mention whether they have trained the model with validating data or not. So, to give a clear understanding of the proposed work, we created two datasets, namely Dataset-1 and Dataset-2. In the Dataset-1, annotated frames for every class are divided into training-set and test-set. Sixty percent of each class is taken as training-set and rest 40% as test-set. The Dataset-2 contains training-set, validation-set, and test-set. Each density class is divided into 40% of training, 20% of validation, and 40% of testing. The following Table 4 and Table 5 show the division of datasets into "Dataset-1" and "Dataset-2".

4.2 Initializing network parameters

The model is fine-tuned by setting the parameters and hyperparameters as per the following values. The parameters of the model, such as learning rate and weights, are initialized to 0.01 and default values, respectively. The hyperparameters such as batch normalization with momentum, number of iterations, L_2 regularization are initialized to 0.95, 500, and 0.01, respectively. For Adam $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The batch sizes of 64,128, 256, 64 used for PETS-2009, UCSD-Ped1, UCSD-Ped2, UMN, respectively.

4.3 Performance metrics

As the proposed work is a multi-class classification problem, so we used macro-average performance metrices for performance analysis. For a binary classification problem, the structure of the confusion matrix is shown in below Table 6.

Using the confusion matrix, we can draw the following performance metrics for individual classes $i|_{i=0,1,2,3,4}$. Here, 0, 1, 2, 3, 4 stands for VL, L, M, H, VH.

$$\text{Recall (Ri)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{24}$$

$$\text{Precision (Pi)} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{25}$$

$$\text{False Positive Rate}\left(\text{FPR}_i\right) = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{26}$$

$$\text{Specificity}\left(\text{S}_i\right) = 1 - \text{FPR}_i \tag{27}$$

$$\text{F1} - \text{Score}\left(\text{F1}_i\right) = \frac{2 \times P_i \times R_i}{P_i + R_i} \tag{28}$$

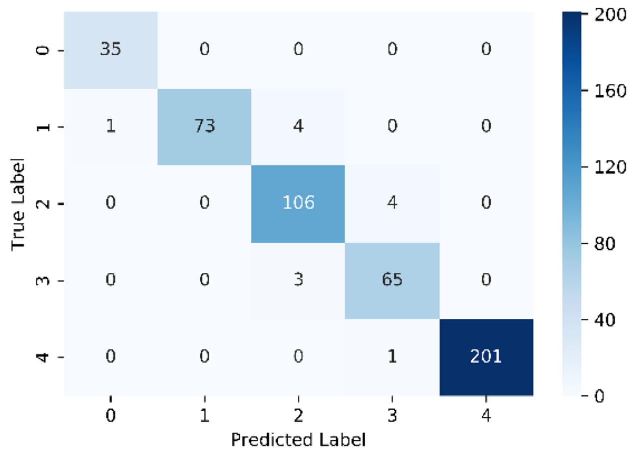

Fig. 5 Confusion Matrix-Heatmap of TIS-MCMS-CNN for PETS-2009 of Dataset-1

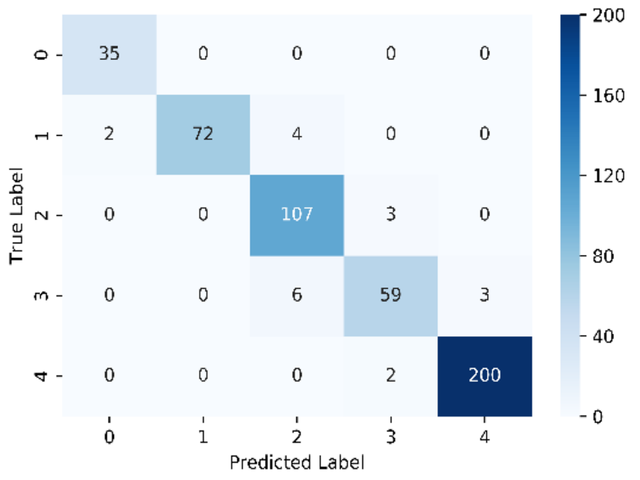

Fig. 6 Confusion Matrix-Heatmap of TIS-MCMS-CNN for PETS-2009 of Dataset-2

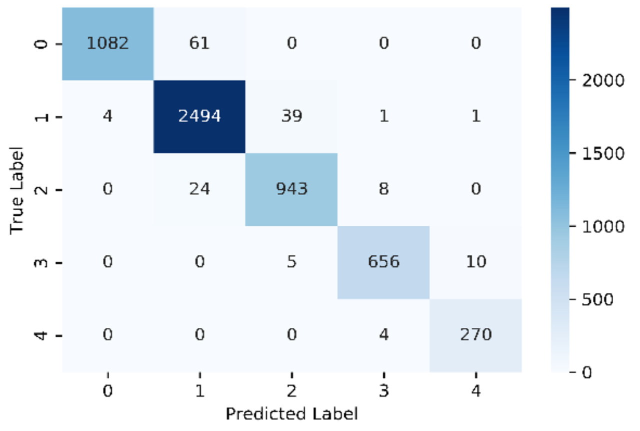

Fig. 7 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UCSD-Ped1 of Dataset-1

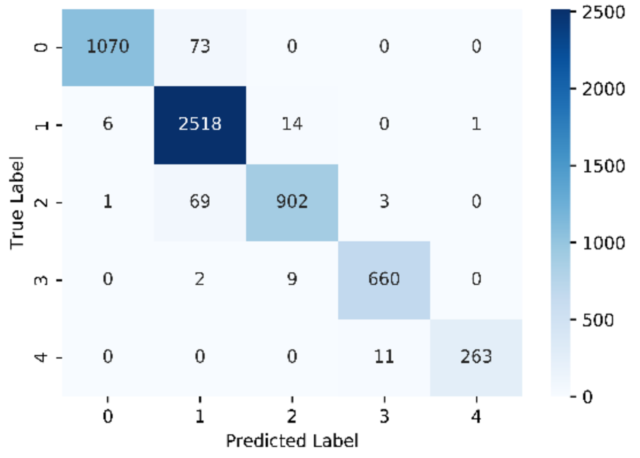

Fig. 8 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UCSD-Ped1 of Dataset-2

Now, the global performance metric for the multi-class classification can be calculated by taking the macro-average of performance metric of individual class. The following equations show macro-average performance metric.

$$\text{Recall}(R) = \frac{\sum_{i=1}^{5} R_i}{5} \tag{29}$$

$$\text{Precision}(P) = \frac{\sum_{i=1}^{5} P_i}{5} \tag{30}$$

$$\text{False Positive Rate(FPR)} = \frac{\sum_{i}^{5} \text{FPR}_i}{5} \tag{31}$$

$$\text{Specificity}(S) = 1 - \text{FP Rate} \tag{32}$$

$$\text{F1} - \text{Score} = \frac{\sum_{i=1}^{5} \text{F1}_i}{5}. \tag{33}$$

The accuracy and error rate can be calculated using the following equations.

$$\text{Accuracy} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{34}$$

$$\text{Error Rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{35}$$

4.4 Results analysis

We performed the experiments by implementing the proposed TIS-MCMS-CNN as well as three state-of-the-art techniques proposed by Fu et al. [10], Alzalani et al. [19]

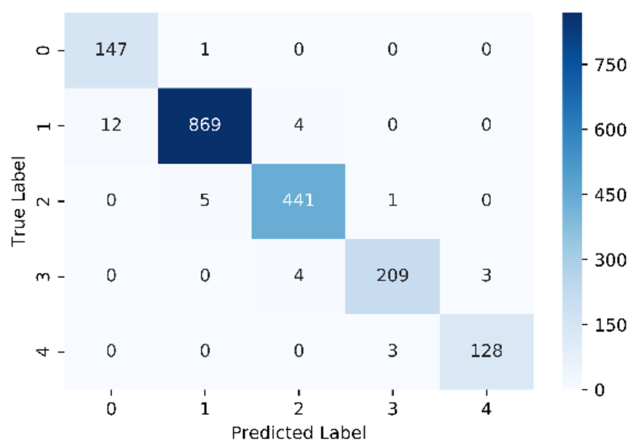

Fig. 9 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UCSD-Ped2 of Dataset-1

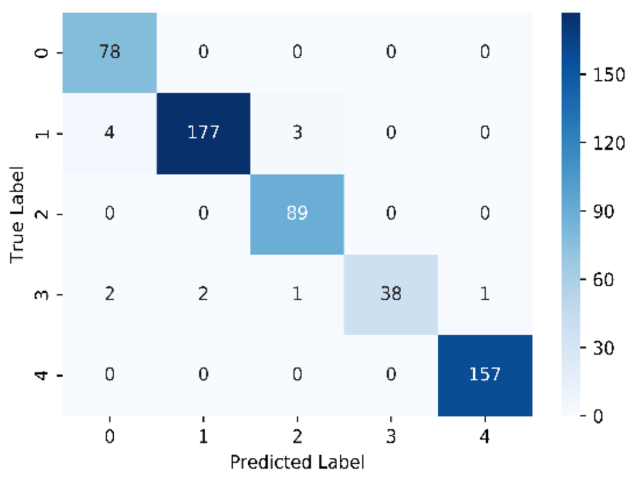

Fig. 12 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza2 of Dataset-2

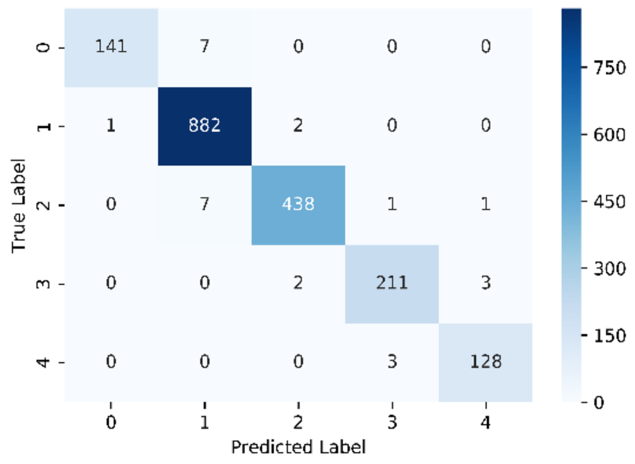

Fig. 10 Confusion Matrix-Heatmap of TIS-MCMS-CNN of UCSD-Ped2 of Dataset-2

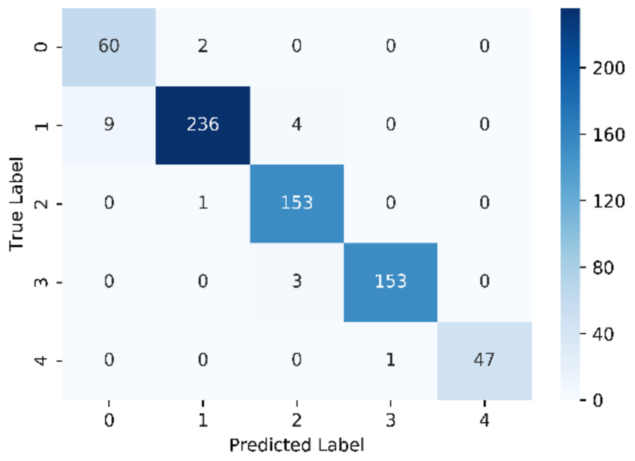

Fig. 13 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza2 of Dataset-1

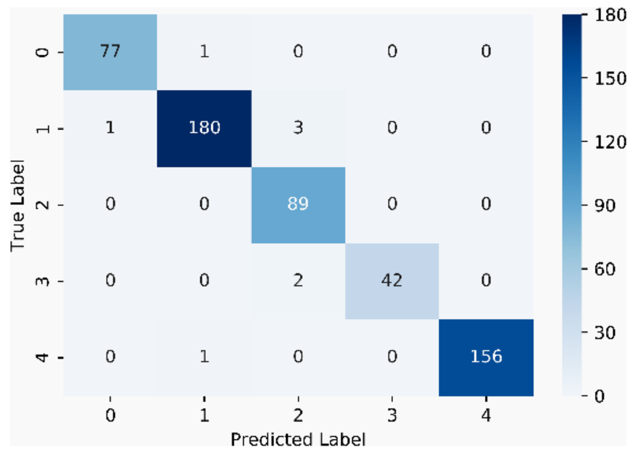

Fig. 11 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza1 of Dataset-1

and Kim et al. [20] on the prepared datasets like Dataset-1 and Dataset-2. Figures 5, 6, 7, 8, 9, 10, 11, 12, 13 and 14 show the confusion matrix heatmaps of the proposed model for the Dataset-1 and Dataset-2.

4.4.1 PETS-2009

Table 7 shows the results of different approaches using Dataset-1 and Dataset-2 of PETS-2009. From Table 7, it can be noticed that the proposed architecture provides better results as compared with the state-of-the-art techniques. Irrespective of challenges such as lighting changes, dynamic crowd shape, and occlusion exists in the dataset, the accuracy of the proposed model for the Dataset-1 is 96.97%, with only 15 misclassified samples. The accuracy for the Dataset-2 is 95.94%, with only 20 misclassified samples. It proves that

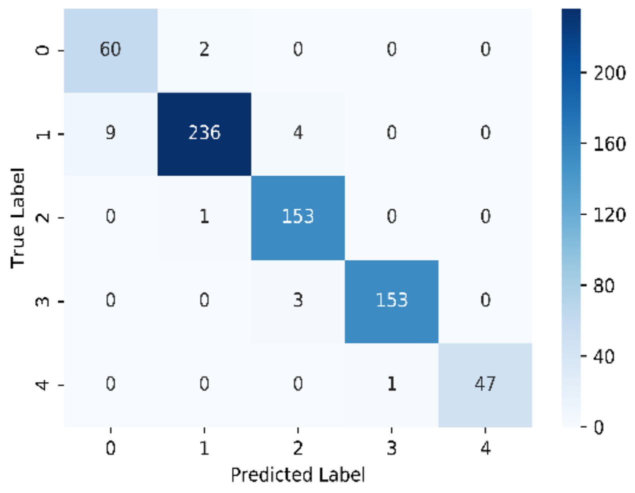

Fig. 14 Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza2 of Dataset-2

the proposed architecture can extract efficient features irrespective of challenging situations and outperforms the MS-CNN [10], CLBP [19], and MLP [20].

4.4.2 UCSD-Ped1

The proposed method achieves the highest accuracy of 97.21% and 96.63% for Dataset-1 and Dataset-2 of UCSD-Ped1, respectively, as compared with the state-of-the-art techniques. Out of 5602 test cases, the misclassified samples are 156 and 189 for Dataset-1 and Dataset-2, respectively. The proposed method shows better performance in terms of Error, Recall, Specificity, Precision, False positive rate, and F1-measures. Table 8 shows details of the achieved performance measurements for different approaches.

4.4.3 UCSD-Ped2

Out of 1427 test cases of UCSD-Ped2, the proposed model results in misclassified samples of 26 and 21, with the accuracy of 98.19% and 98.52% for Dataset-1 and Dataset-2, respectively.

The performance analysis on UCSD-Ped2 is given in Table 9, and it can be noticed that the proposed architecture performs better than other techniques in terms of Error, Recall, Specificity, Precision, False positive rate, and F1-measures. Thus, it capably handles the challenging situations that exist in the dataset.

Table 7 Performance Analysis of several approaches using Dataset PETS-2009

Dataset	Approaches	Performance metrics						
		Accuracy	Error	*R*	*S*	*P*	FPR	F1_score
Dataset-1	MS-CNN [10]	0.9412	0.0588	0.9338	0.9856	0.9298	0.0144	0.9309
	CLBP [19]	0.929	0.071	0.9104	0.9821	0.9227	0.0179	0.915
	MLP [20]	0.5071	0.4929	0.4082	0.8635	0.32	0.1365	0.292
	TIS- MCMS-CNN	**0.9697**	**0.0303**	**0.9667**	**0.9926**	**0.9624**	**0.0074**	**0.9644**
Dataset-2	MS-CNN [10]	0.9329	0.0671	0.9104	0.9826	0.9285	0.0174	0.9183
	CLBP [19]	0.929	0.071	0.9085	0.9819	0.9242	0.0181	0.9145
	MLP [20]	0.501	0.499	0.329	0.858	0.2665	0.142	0.2896
	TIS- MCMS-CNN	**0.9594**	**0.0406**	**0.9507**	**0.9895**	**0.9535**	**0.0105**	**0.9513**

Bold values represent higher values

Table 8 Performance analysis of several approaches on UCSD-Ped1

Dataset	Approaches	Performance metrics						
		Accuracy	Error	*R*	*S*	*P*	FPR	F1_score
Dataset-1	MS-CNN [10]	0.9509	0.0491	0.9536	0.9849	0.9348	0.0151	0.9411
	CLBP [19]	0.9135	0.0865	0.8925	0.9737	0.9224	0.0263	0.9062
	MLP [20]	47.32	0.5268	0.2654	0.822	0.3129	0.178	0.248
	TIS- MCMS-CNN	**0.9721**	**0.0279**	**0.9719**	**0.9915**	**0.9727**	**0.0085**	**0.9721**
Dataset-2	MS-CNN [10]	0.9309	0.0691	0.9269	0.9783	0.9459	0.0217	0.9359
	CLBP [19]	0.9135	0.0865	0.8925	0.9737	0.9224	0.0263	0.9062
	MLP [20]	0.4716	0.5284	0.249	0.8192	0.2961	0.1808	0.1936
	TIS- MCMS-CNN	**0.9663**	**0.0337**	**0.9593**	**0.9887**	**0.978**	**0.0113**	**0.9682**

Bold values represent higher values

Table 9 Performance analysis of several approaches on UCSD-Ped2

Dataset	Approaches	Performance metrics						
		Accuracy	Error	*R*	*S*	*P*	FPR	F1_score
Dataset-1	MS-CNN [10]	0.9217	0.0783	0.8943	0.9756	0.9231	0.0244	0.9075
	CLBP [19]	0.8955	0.1045	0.8467	0.9669	0.904	0.0331	0.8703
	MLP [20]	50.95	0.4905	0.256	0.8143	0.2054	0.1857	0.3552
	TIS- MCMS-CNN	**0.9819**	**0.0181**	**0.9813**	**0.9953**	**0.9716**	**0.0047**	**0.9762**
Dataset-2	MS-CNN [10]	0.8747	0.1253	0.8349	0.9633	0.8789	0.0367	0.8511
	CLBP [19]	0.902	0.098	0.8608	0.9684	0.9138	0.0316	0.883
	MLP [20]	0.5014	0.4986	0.2149	0.812	0.2043	0.188	0.161
	TIS- MCMS-CNN	**0.9852**	**0.0148**	**0.9766**	**0.9954**	**0.9839**	**0.0046**	**0.9801**

Bold values represent higher values

Table 10 Performance analysis of several approaches on UMN-Plaza1

Dataset	Approaches	Performance metrics						
		Accuracy	Error	*R*	*S*	*P*	FPR	F1_score
Dataset-1	MS-CNN [10]	0.9511	0.0489	0.9521	0.988	0.936	0.012	0.9435
	CLBP [19]	0.9837	0.0163	0.9856	0.996	0.9813	0.004	0.9832
	MLP [20]	0.7717	0.2283	0.7068	0.9423	0.7564	0.0577	0.6607
	TIS- MCMS-CNN	**0.9855**	**0.0145**	**0.9827**	**0.9963**	**0.9846**	**0.0037**	**0.9834**
Dataset-2	MS-CNN [10]	0.9511	0.0489	0.9521	0.988	0.936	0.012	0.9435
	CLBP [19]	0.9837	0.0163	0.9856	0.996	0.9813	0.004	0.9832
	MLP [20]	0.71	0.29	0.6131	0.9232	0.5722	0.0768	0.588
	TIS- MCMS-CNN	**0.9764**	**0.0236**	**0.9651**	**0.9941**	**0.9736**	**0.0059**	**0.968**

Bold values represent higher values

Table 11 Performance analysis of several approaches on UMN-Plaza2

Dataset	Approaches	Performance Metrics						
		Accuracy	Error	*R*	*S*	*P*	FPR	F1_score
Dataset-1	MS-CNN [10]	0.8879	0.1121	0.8884	0.9666	0.9226	0.0334	0.8975
	CLBP [19]	0.7653	0.2347	0.8715	0.9391	0.8876	0.0609	0.837
	MLP [20]	0.728	0.272	0.577	0.9225	0.6016	0.0774	0.56
	TIS- MCMS-CNN	**0.9701**	**0.0299**	**0.9738**	**0.9925**	**0.9614**	**0.0075**	**0.9669**
Dataset-2	MS-CNN [10]	0.9178	0.0822	0.9245	0.9749	0.9458	0.0251	0.9314
	CLBP [19]	0.7638	0.2362	0.8702	0.9387	0.8873	0.0613	0.8361
	MLP [20]	0.725	0.275	0.5846	0.9218	0.591	0.0782	0.5681
	TIS- MCMS-CNN	**0.9701**	**0.0299**	**0.9738**	**0.9925**	**0.9614**	**0.0075**	**0.9669**

Bold values represent higher values

4.4.4 UMN-Plaza1 and Plaza2

The proposed method achieves an accuracy of 98.55% and 97.64% with misclassified samples of 8 and 13 for Dataset-1 and Dataset-2 of UMN-Plaza1, respectively. Similarly, for UMN-Plaza2, the proposed method again performs well in terms of accuracy and other measures, as shown in Tables 10 and 11. Although CLBP [19] performs better in Dataset-2, it takes much time to process a frame. The time analysis for all the approaches is discussed in Section-4.6. Figs. 15 and 16 show graphical representations of the performance comparison of several methods for Dataset-1 and Dataset-2.

4.5 Ablation study

The ablation study is done to show the impact of each stream that is spatial and motion-stream individually. The results are shown in Table 12. The accuracies of two streams are very low as compared with the proposed architecture. It can be concluded from Table 12 that neither spatial-stream nor the

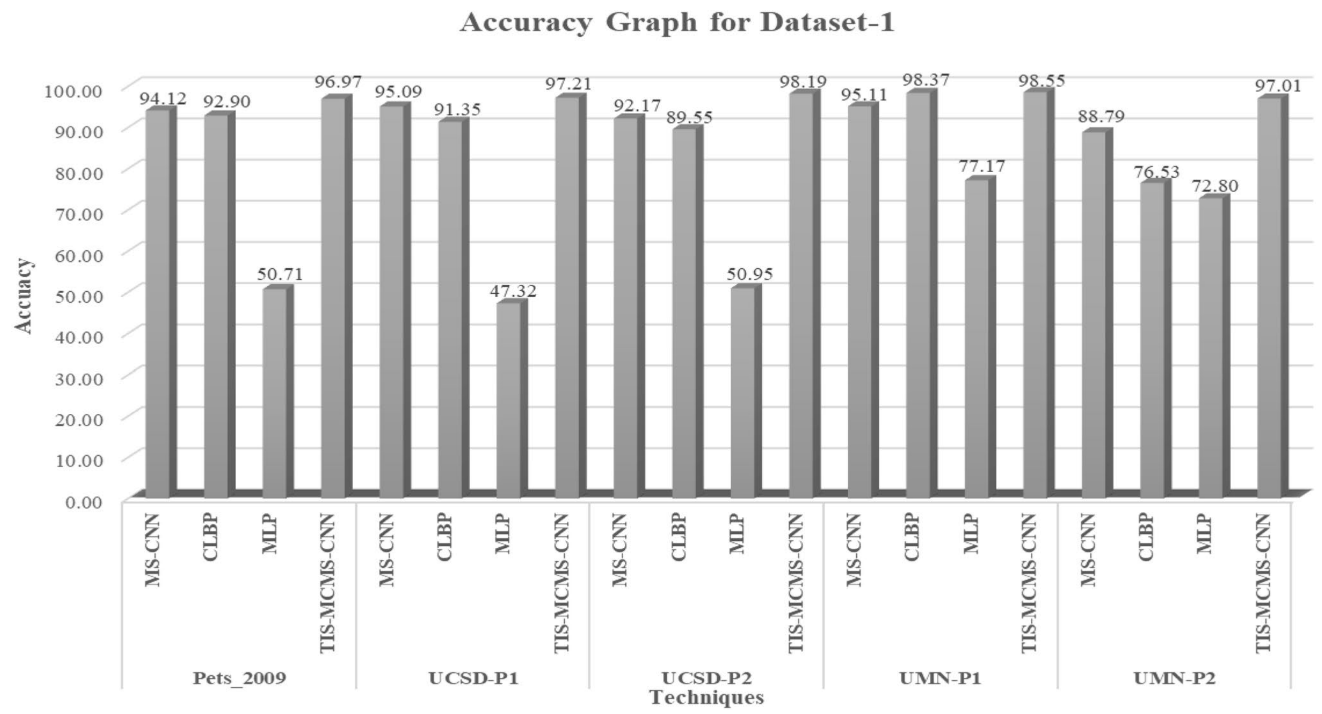

Fig. 15 Comparison of accuracies of several approaches for “Dataset-1”

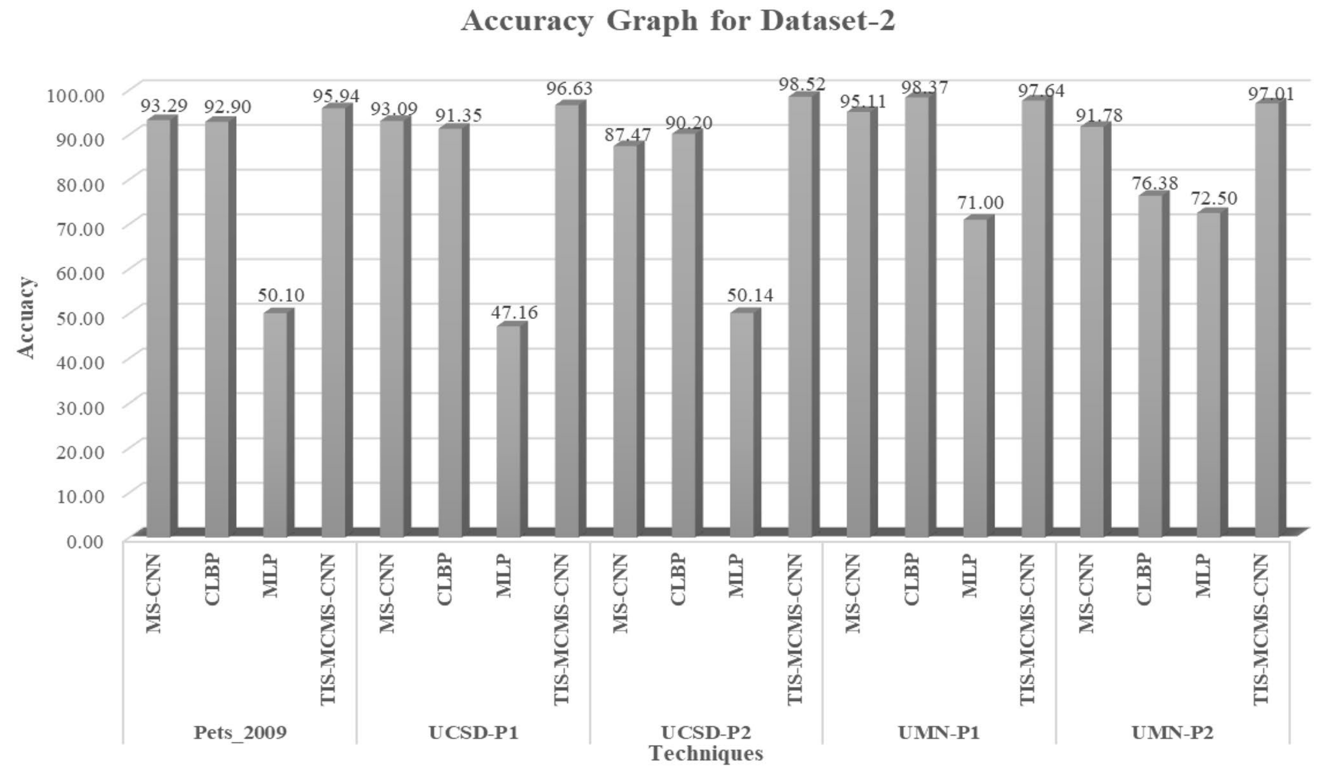

Fig. 16 Comparison of accuracies of several approaches for “Dataset-2”

Table 12 Performance analysis of Ablation Study on Different Datasets

Datasets	Approaches	Performance metrics						
		Accuracy	Error	*R*	*S*	*P*	FPR	F1_score
PETS-2009								
Dataset-1	Spatial-Stream	0.9574	0.0426	0.9412	0.9893	0.9537	0.0107	0.9454
	Motion-Stream	0.9087	0.0913	0.896	0.9758	0.9026	0.0242	0.8974
	Proposed Model	**0.9697**	**0.0303**	**0.9667**	**0.9926**	**0.9624**	**0.0074**	**0.9644**
Dataset-1	Spatial-Stream	0.9493	0.0507	0.9347	0.9876	0.9307	0.0124	0.9308
	Motion-Stream	0.8966	0.1034	0.8851	0.9739	0.8835	0.0261	0.8824
	Proposed Model	**0.9594**	**0.0406**	**0.9507**	**0.9895**	**0.9535**	**0.0105**	**0.9513**
UCSD- Ped1								
Dataset-1	Spatial-Stream	0.9529	0.0471	0.9107	0.9858	0.9578	0.0142	0.9297
	Motion-Stream	0.8954	0.1046	0.8792	0.9681	0.9043	0.0319	0.8904
	Proposed Model	**0.9721**	**0.0279**	**0.9719**	**0.9915**	**0.9727**	**0.0085**	**0.9721**
Dataset-1	Spatial-Stream	0.9531	0.0469	0.9506	0.9855	0.9598	0.0145	0.9549
	Motion-Stream	0.8563	0.1437	0.8172	0.9568	0.8644	0.0432	0.8366
	Proposed Model	**0.9663**	**0.0337**	**0.9593**	**0.9887**	**0.978**	**0.0113**	**0.9682**
UCSD- Ped2								
Dataset-1	Spatial-Stream	0.9535	0.0465	0.9333	0.9882	0.9361	0.0118	0.9297
	Motion-Stream	0.9064	0.0936	0.8577	0.9731	0.8867	0.0269	0.8677
	Proposed Model	**0.9819**	**0.0181**	**0.9813**	**0.9953**	**0.9716**	**0.0047**	**0.9762**
Dataset-1	Spatial-Stream	0.9819	0.0181	0.9778	0.995	0.9753	0.005	0.9764
	Motion-Stream	0.8862	0.1138	0.8445	0.9668	0.8507	0.0332	0.8462
	Proposed Model	**0.9852**	**0.0148**	**0.9766**	**0.9954**	**0.9839**	**0.0046**	**0.9801**
UMN-Plaza1								
Dataset-1	Spatial-Stream	0.9438	0.0562	0.8734	0.9859	0.9455	0.0141	0.8829
	Motion-Stream	0.9674	0.0326	0.9613	0.9921	0.9559	0.0079	0.958
	Proposed Model	**0.9855**	**0.0145**	**0.9827**	**0.9963**	**0.9846**	**0.0037**	**0.9834**
Dataset-1	Spatial-Stream	0.8877	0.1123	0.8227	0.9744	0.8658	0.0256	0.8004
	Motion-Stream	0.9638	0.0362	0.9546	0.9912	0.9512	0.0088	0.9522
	Proposed Model	**0.9764**	**0.0236**	**0.9651**	**0.9941**	**0.9736**	**0.0059**	**0.968**
UMN-Plaza2								
Dataset-1	Spatial-Stream	0.9118	0.0882	0.9423	0.9775	0.9093	0.0225	0.9208
	Motion-Stream	0.8759	0.1241	0.7729	0.9612	0.8441	0.0388	0.7599
	Proposed Model	**0.9701**	**0.0299**	**0.9738**	**0.9925**	**0.9614**	**0.0075**	**0.9669**
Dataset-1	Spatial-Stream	0.8819	0.1181	0.9194	0.9719	0.8742	0.0281	0.8834
	Motion-Stream	0.867	0.133	0.7884	0.9605	0.8396	0.0395	0.8013
	Proposed Model	**0.9701**	**0.0299**	**0.9738**	**0.9925**	**0.9614**	**0.0075**	**0.9669**

Bold values represent higher values

Table 13 Test frames processing time of several approaches

Frame-rate Vs approaches	Approaches				
	MS-CNN [10]	MLP [18]	CLBP [20]	Spatial-Stream	TIS-MCMS-CNN
Average execution time (frames/s)	65	25	0.143	43.49	29.45

motion-stream solely capable of capturing discriminant features. Generally, cluttered background, dynamic crowd shape, and occlusion affect the spatial-stream, and static objects like humans affect the motion-stream. Moreover, when we combine these two features, the performance increases.

4.6 Time analysis

We also calculated the average time required to process the test frames. It is found that the proposed approach achieves around 29.45 (≈30) frames per second. Although MS-MCNN performs better still, the proposed model can provide results in real-time. The following Table 13 shows average frames per second achieved for test cases by different approaches.

5 Conclusion and future work

The proposed architecture (TIS-MCMS-CNN) implements a global crowd congestion-level analysis. The two streams of TIS-MCMS-CNN extracts both spatial and temporal features from the resized frame and motion magnitude map, respectively. The motion magnitude map is obtained by using the Lucas–Kanade [34] optical flow. Other optical flow algorithms can be used, but the Lucas–Kanade [34] approach provides noise-free temporal information between consecutive frames. The TIS-MCMS-CNN can extract features invariant to perspective change and scene change. To measure the efficiency of the model, we demonstrated experiments on three publicly available datasets, namely PETS-2009 [24], UCSD [25], UMN [26]. These datasets contain some of the challenging situations such as illumination changes, occlusion, perspective change, and camera jitter. We compared the performance of the proposed model with MS-CNN [10], CLBP [19], and MLP [20]. The proposed model outperforms the state-of-the-art techniques. The accuracy proves that the model is robust and can capture discriminant features even in the presence of some challenging situations. The architecture processes an average of nearly 30 test frames per second, and hence, it is applicable in real-time. The model could not classify some samples in which the transition between different congestion classes occurs. In our future work, we focus on developing a real-time implementation of a deep graphical model and applying image understanding tasks [32, 33] in the local-CCA.

References

1. Jiang, X., Xiao, Z., Zhang, B., et al.: Crowd counting and density estimation by trellis encoder-decoder networks. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp 6126–6135 (2019).https://doi.org/10.1109/CVPR.2019.00629
2. Chen, X.H., Lai, J.H.: Detecting abnormal crowd behaviors based on the div-curl characteristics of flow fields. Pattern Recognit. **88**, 342–355 (2019). https://doi.org/10.1016/j.patcog.2018.11.023
3. Wei, X., Du, J., Xue, Z., et al.: A very deep two-stream network for crowd type recognition. Neurocomputing (2019). https://doi.org/10.1016/j.neucom.2018.10.106
4. Vahora, S.A., Chauhan, N.C.: Deep neural network model for group activity recognition using contextual relationship. Eng. Sci. Technol. Int. J. **22**, 47–54 (2019). https://doi.org/10.1016/j.jestch.2018.08.010
5. Jing, S., Chen, C.L., Kai Kang, X.W.: Slicing convolutional neural network for crowd video understanding. In: Proc IEEE Conf Comput Vis Pattern Recognition 5620–5628 (2016)
6. Xiong, G., Cheng, J., Wu, X., et al.: An energy model approach to people counting for abnormal crowd behavior detection. Neurocomputing **83**, 121–135 (2012). https://doi.org/10.1016/j.neucom.2011.12.007
7. Lazaridis, L., Dimou, A., Daras, P.: Abnormal behavior detection in crowded scenes using density heatmaps and optical flow. In: Eur Signal Process Conf 2018-September, pp 2060–2064. (2018) https://doi.org/10.23919/EUSIPCO.2018.8553620
8. Huang, L., Chen, T., Wang, Y., Yuan, H.: Congestion detection of pedestrians using the velocity entropy: A case study of Love Parade 2010 disaster. Phys. A Stat. Mech. Appl. **440**, 200–209 (2015). https://doi.org/10.1016/j.physa.2015.08.013
9. Polus, A., Schofer, J.L., Ushpiz, A.: Pedestrian flow and level of service. J Transp Eng **109**, 46–56 (1983). https://doi.org/10.1061/(ASCE)0733-947X(1983)109:1(46)
10. Fu, M., Xu, P., Li, X., et al.: Fast crowd density estimation with convolutional neural networks. Eng. Appl. Artif. Intell. **43**, 81–88 (2015). https://doi.org/10.1016/j.engappai.2015.04.006
11. Marana, A.N., Velastin, S.A., Costa, L.F., Lotufo, R.A.: Automatic estimation of crowd density using texture. Saf. Sci. **28**, 165–175 (1998). https://doi.org/10.1016/S0925-7535(97)00081-7
12. Marana, A.N., da Costa, L.F., Lotufo, R.A., Velastin, S.A.: Estimating crowd density with Minkowski fractal dimension. ICASSP IEEE Int. Conf. Acoust. Speech Signal Process Proc **6**, 3521–3524 (1999). https://doi.org/10.1109/icassp.1999.757602
13. Rahmalan, H., Nixon, M.S., Carter, J.N.: On crowd density estimation for surveillance. 540–545. (2008) https://doi.org/10.1049/ic:20060360
14. Marana, A.N., Cavenaghi, M.A., Ulson, R.S., Drumond, F.L.: Real-Time Crowd Density Estimation Using Images. Springer, Berlin (2005)
15. Su, H., Yang, H., Zheng, S.: The large-scale crowd density estimation based on effective region feature extraction method. In: Kimmel, R., Klette, R., Sugimoto, A. (eds) Computer Vision – ACCV 2010. ACCV 2010. Lecture Notes in Computer Science, vol 6494. Springer, Berlin, Heidelberg, pp 302–313 (2011). https://doi.org/10.1007/978-3-642-19318-7_24
16. Ma, W., Huang, L., Liu, C.: Crowd density analysis using co-occurrence texture features. In: Proceeding-5th Int Conf Comput Sci Converg Inf Technol ICCIT 2010 170–175. (2010) https://doi.org/10.1109/ICCIT.2010.5711051
17. Wang, Z., Liu, H., Qian, Y., Xu, T.: Crowd density estimation based on local binary pattern co-occurrence matrix. In: Proc 2012 IEEE Int Conf Multimed Expo Work ICMEW 2012 372–377. (2012) https://doi.org/10.1109/ICMEW.2012.71
18. Fradi, H., Dugelay, J.L.: A new multiclass SVM algorithm and its application to crowd density analysis using LBP features. in: 2013 IEEE Int Conf Image Process ICIP 2013-Proc 4554–4558. (2013) https://doi.org/10.1109/ICIP.2013.6738938
19. Alanazi, A.A., Bilal, M., Engineering, S.: Crowd Density Estimation Using Novel Feature Descriptor. (2019) arXiv:190505891
20. Kim, G.: Estimation of Crowd Density in Public Areas Based on Neural Network. KSII Trans Internet Inf Syst **6**, 2170–2190 (2012). https://doi.org/10.3837/tiis.2012.09.011
21. Yang, H., Su, H., Zheng, S., et al.: The large-scale crowd density estimation based on sparse spatiotemporal local binary pattern. Proc IEEE Int. Conf. Multimed. Expo. (2011). https://doi.org/10.1109/ICME.2011.6012156
22. Pu, S., Song, T., Zhang, Y., Xie, D.: Estimation of crowd density in surveillance scenes based on deep convolutional neural network. Proc. Comput. Sci. **111**, 154–159 (2017). https://doi.org/10.1016/j.procs.2017.06.022
23. Zhang, Y., Zhou, D., Chen, S., et al.: Single-image crowd counting via multi-column convolutional neural network. Proc IEEE Conf. Comput. Vis Pattern Recognit. (2016). https://doi.org/10.1002/slct.201701956
24. PETS 2009 Benchmark Data.: http://www.cvg.reading.ac.uk/PETS2009/a.html#s111. Accessed 31 Jul 2019
25. UCSD Anomaly Detection Dataset.: http://www.svcl.ucsd.edu/projects/anomaly/dataset.htm. Accessed 24 May 2019
26. Monitoring Human Activity.: http://mha.cs.umn.edu/proj_recognition.shtml#crowd_count. Accessed 27 May 2019

27. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide-baseline stereo from maximally stable extremal regions. Image Vis. Comput. **22**, 761–767 (2004). https://doi.org/10.1016/j.imavis.2004.02.006
28. Lamba, S., Nain, N.: A large scale crowd density classification using spatio-temporal local binary pattern. In: Proc-13th Int Conf Signal-Image Technol Internet-Based Syst SITIS 2017 2018-January, pp 296–302. (2018) https://doi.org/10.1109/SITIS.2017.57
29. Mikolajczyk, K., Tuytelaars, T., Schmid, C., et al.: A comparison of affine region detectors. Int J Comput Vis **65**, 43–72 (2005). https://doi.org/10.1007/s11263-005-3848-x
30. Szegedy, C., Liu, W., Jia, Y., et al.: Going deeper with convolutions. Proc IEEE Conf. Comput. Vis Pattern Recognit. (2015). https://doi.org/10.1002/jctb.4820
31. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd Int Conf Learn Represent ICLR 2015 - Conf Track Proc, pp 1–14 (2015)
32. Li, Z., Tang, J.: Weakly supervised deep metric learning for community-contributed image retrieval. IEEE Trans. Multimed. **17**, 1989–1999 (2015). https://doi.org/10.1109/TMM.2015.2477035
33. Li, Z., Tang, J., Mei, T.: Deep collaborative embedding for social image understanding. IEEE Trans. Pattern Anal. Mach. Intell. (2018). https://doi.org/10.1109/TPAMI.2018.2852750
34. Bruce, L.D., Kanade, T.: An iterative image registration technique with an application to stereo vision. Proceedings DARPA image Understanding workshop (1981):121430.
35. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986). https://doi.org/10.1038/323533a0
36. Kingma, D. P., Ba, J. L.: Adam: A method for stochastic optimization. 3rd Int Conf Learn Represent ICLR 2015 - Conf Track Proc 1–15 (2015)