CHAPTER 4
A REAL TIME TWO INPUT STREAM MULTI COLUMN MULTISCALE CNN FOR EFFICIENT CROWD CONGESTION-LEVEL ANALYSIS

4.1 Introduction

The CCA provides congestion information in a crowd scene, which helps with crowd disaster management. The CCA can be implemented at the global (Frame-level) or local level (Patch-Level). 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. However, 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. [178] or manually defining classes based on density and crowd flow information [115]. For example, the congestion classes could be free-flow, restricted-flow, jammed-flow, and dense-flow [178] or very-low (VL), low (L), medium (M), and high (H), and very high (VH) [115]. The existing conventional approaches extract spatial (shape, spectral, texture) [40], [114], [116]–[122] or spatial-temporal texture features [123], [124] to solve the CCA as a multiclass classification problem. These methods lack in extracting fine-grained features, resulting in a high misclassification rate, and are computationally very expensive. The existing deep-learning frameworks utilize CNN architecture [125], [178] to extract spatial features to solve the CCA. This chapter proposes a work based on two intuitions (i) extracting only spatial features will not increase the accuracy since the crowd scene is affected by cluttered background, lighting change, varying crowd densities, and 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 them. Moreover, (ii) the single-column CNN [125], [178] for CCA cannot capture features invariant to perspective change or scene change, but multi-column CNN [27] with different kernel sizes is capable of extracting invariant features. Hence based on these two intuitions, a two-input stream multi-column multi-stage CNN (TIS-MCMS-CNN) is proposed to solve CCA in real-time, which is discussed in a subsequent section.

4.2 The Proposed Model: A Real time Two Input Stream Multi Column Multiscale CNN for Efficient Crowd Congestion-level Analysis

The proposed TIS-MCMS-CNN consists of two input streams of a multi-column multi-stage convolutional neural network for the frame-level CCL. Figure 4.1 and Figure 4.2 shows the overall architecture and the detailed architecture of the TIS-MCMS-CNN, respectively.

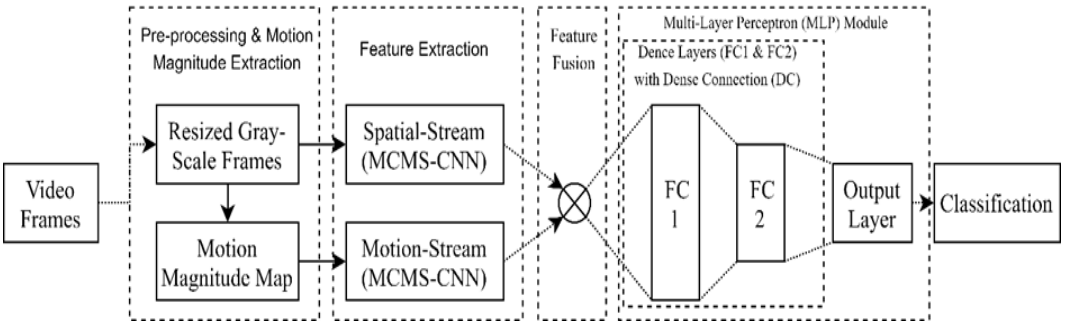

Figure 4.1: Overall architecture of the proposed model

The following sub-sections explain details of the proposed model and it's working principle.

- Network Architecture.
- Pre-processing and Motion Magnitude Map Extraction.
- Problem Formulation and Learning Algorithm.
- Precaution to handle overfitting.

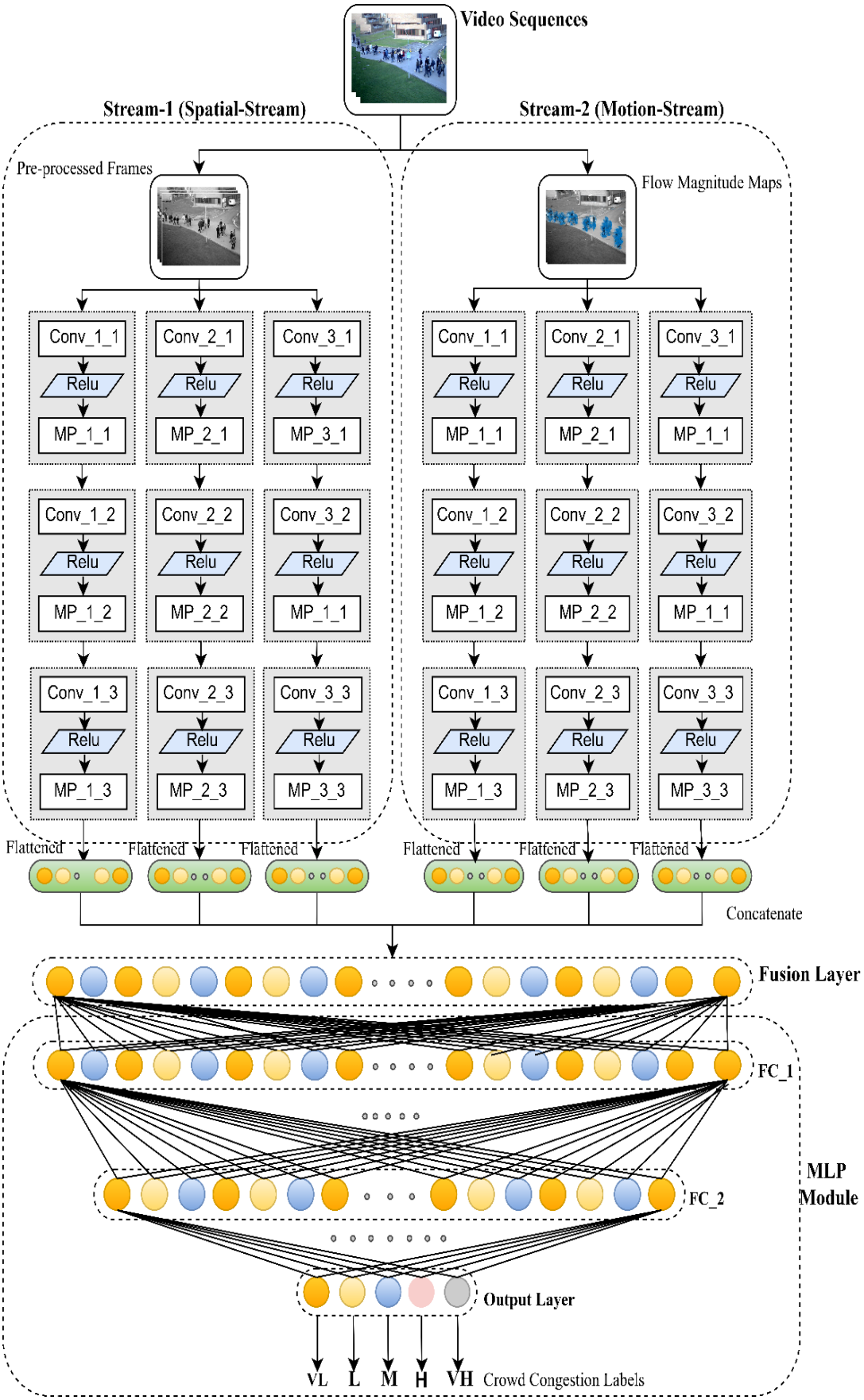

Figure 4.2: Detail architecture of the model TIS-MCMS-CNN

Table 4.1: 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	

4.2.1 Network Architecture

According to Figure 4.2, 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 Figure 4.2) are 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 fields or kernel size. The details of the layer information are given in Table 4.1. 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 k^{th} neuron at level l can be calculated using the following Equation 4.1.

$$ReLU\left(z_{in_k}\right)_l = max\{0, z_{in_k}\} \tag{4.1}$$

, where $\left(z_{in_k}\right)$ is the weighted sum of the information transmitted from neurons of level *l-1* to the k^{th} 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: 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 like 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 Equation 4.2 shows the SoftMax activation for $the\ p^{th}$ neuron of the seventh layer (output) of our proposed model.

$$y_{7_{p_{out}}} = SoftMax\left(\left(z_{in_p}\right)_7\right) = \frac{e^{\left(z_{in_p}\right)_7}}{\sum_{p=1}^{5} e^{\left(z_{in_p}\right)_7}}, for\ p = 1,2,3,4,5 \quad (4.2)$$

, where $\left(z_{in_p}\right)_7$ is the weighted information transmitted from the sixth layer to the pth neuron of the output layer (seventh layer).

4.2.2 Pre-processing and Motion Magnitude Map Extraction

In the pre-processing stage, by following the work of Fu *et al.* [115]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 $VF = \{vf_1, vf_2, \ldots\ldots, vf_T\}$ and $GF = \{gf_1, gf_2, \ldots\ldots, gf_T\}$ respectively, where T is the total number of frames. Equation 4.3 is used to convert the colour frames into grayscale images.

$$gf_i = 0.299 \times R(vf_i) + 0.587 \times G(vf_i) + 0.114 \times B(vf_i), \forall i = 1,2, \ldots, T \quad (4.3)$$

, 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.* [115]. Then, the motion magnitude map of the resized frames is obtained by applying the Lucas-Kanade [179] 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.

$$gf_x \times u + gf_y \times v + gf_d = 0 \tag{4.4}$$

, $gf_x \; and \; gf_y$ are the spatial derivatives of *the* d^{th} frame, gf_T is the temporal derivative of $the\ d^{th}\ frame,$ and u, v are the horizontal and vertical optical flow of that frame, respectively. The flow magnitude can be obtained by solving the Equation 4.5.

$$Mag(x, y, d) = \sqrt{u(x, y)^2 + v(x, y)^2} \tag{4.5}$$

, where $Mag(x, y, d)$ refers to the motion magnitude map for the d^{th} frame. The detailed description and solution of Equation 4.4 and Equation 4.5 can be found in [179]. Let the motion magnitude maps for the set *GF* is denoted as set $MF = \{mf_1, mf_2, \ldots\ldots, mf_T\}$. The resized grayscale frame set *GF* and the motion magnitude set *MF* are inputted to the first stream i.e., the 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.

4.2.3 Problem Formulation & Learning Mechanism

Let GF and MF are divided into N number of batches of samples. Let tuple $S_{t_x} = < GF_{t_x}, MF_{t_x} >$ represents pre-processed gray frames and corresponding motion magnitude maps for the t^{th} 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 t^{th} 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 i^{th} layer for the j^{th} column of k^{th} stream is denoted as,

$$\left[f_{i,k}^{j}\right]^{S_{t_x}} = \left[CONV\left(\theta_{C_{i,k}}^{j}, \left[fm_{i-1,k}^{j}\right]^{T}\right)\right]^{S_{t_x}} \tag{4.6}$$

, where θ_C represents parameters for two-stream CNNs and $\theta_{C_{i,k}}^{j} = \left[W_{i1,k}^{j}, W_{i2,k}^{j}, \ldots\ldots, W_{iM,k}^{j}\right]$ for i=1,2,3 j=1,2,3 and k=1,2. Each $W_{im,k}^{j}$ represents m^{th} convolution kernel's weight matrix for i^{th} layer of j^{th} column of k^{th} stream and we got m=1,2,…,*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 j^{th} column of $the\ k^{th}$ stream. Not 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 a max pooling layer. The feature map of each stage can be calculated as,

$$\left[fm_{i,k}^{j}\right]^{S_{t_x}} = \left[MP\left(ReLU\left(f_{i,k}^{j}\right)\right)\right]^{S_{t_x}} \tag{4.7}$$

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

$$[Fuse]^{S_{t_x}} = \left[CONCATE\left(fm_{3,k}^{j}\right)\right]^{S_{t_x}} \tag{4.8}$$

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,

$$[y_i]^{S_{tx}} = \left[\theta_{fc_i} \times \left[y_{(i-1)_{out}}, 1\right]^T\right]^{S_{tx}} \quad (4.9)$$

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

$$\left[y_{i_{out}}\right]^{S_{tx}} = [ReLU(y_i)]^{S_{tx}}, for\ i = 5{,}6 \quad (4.10)$$

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,

$$\left[y_{i_{out}}\right]^{S_{tx}} = \left[\cup_{p=0}^{4}\left[SoftMax\left(y_{i_{p_{out}}}\right)\right]\right]^{S_{tx}}, for\ i = 7. \quad (4.11)$$

Let $\emptyset_{Net} = \left[\theta_C, \theta_{fc}\right]$ 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 $L\left(T_p, y_{i_{out}}\right)$, can be calculated by using the Equation 4.12.

$$[L(\emptyset_{Net})]^{S_{tx}} = \left[L\left(T_p, y_{i_{out}}\right)\right]^{S_{tx}} = \left[-\sum_{p=0}^{4} T_p \log y_{7_{p_{out}}}\right]^{S_{tx}} \quad (4.12)$$

, where $T_p|_{p=0,1,2,3,4}$ is the true distribution of the five density classes, namely VL (0), L (1), M (2), H (3), and VH (4) and $y_{7_{out}}$ 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,

$$\underset{\emptyset_{Net}}{argmin}\left[-\sum_{p=0}^{4} T_p \log y_{7_{p_{out}}}\right]^{S_{t_x}} \mid \sum_{p=0}^{4} y_{i_{p_{out}}} = 1 \tag{4.13}$$

By using *Lagrangian* constraint multiplier, the objective function can be reduced to the following form

$$\underset{\emptyset_{Net}}{argmin}\left[-\sum_{p=0}^{4} T_p \log y_{7_{p_{out}}} + \lambda \sum_{p=0}^{4} y_{i_{p_{out}}} - 1\right]^{S_{t_x}} \tag{4.14}$$

, where λ is the *Lagrangian* multiplier. If $\lambda \sum_{p=0}^{4} y_{i_{p_{out}}} - 1$=0, then we will get an absolute minimum. In this work, the L_2 norm as the regularization term and added in the optimization function. So, the loss function could look like as,

$$[\tilde{L}(\emptyset_{Net})]^{S_{t_x}} = \left[L(\emptyset_{Net}) + \frac{\alpha}{2}\|\emptyset_{Net}\|^2\right]^{S_{t_x}} \tag{4.15}$$

, where α is the regularized parameter.

The model trained and optimized using backpropagation [169] with Adam optimizer [170]. 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 patience 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.

4.2.4 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 errors 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.

4.3 Dataset Preparation

For the experiment and results analysis, a CCL dataset is prepared using the publicly available crowd datasets, namely, Pets-2009 S1 View1 [180], UCSD Ped1 and Ped2 [181], UMN Plaza-1 and Plaza-2 [182]. The PETS-2009 [180] dataset is the benchmark dataset for crowd surveillance. The Pets-2009 S1 mainly meant for crowd count and density estimation. It contains three scenarios, like L1, L2, and L3. 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 [181] is a crowd anomaly dataset that provides two sequences in two different scenarios, namely, UCSD-Ped1 and UCSD-Ped2. The Ped1 and Ped2 contain 16000 and 4800 frames, respectively. These datasets contain different challenging conditions like varying lighting conditions, occlusions, camera jitters. The UMN [182] provides benchmark datasets for crowd monitoring. The UMN-Plaza1 and Plaza2 are two surveillance videos recorded in Plaza-1 and Plaza-2 for crowd monitoring.

By following the work of Fu *et al.* [115], each of the frames of these datasets 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 like Very-Low (VL), Low (L), Medium (M), High (H), and Very-High (VH). The detail of the division of crowd density levels is given in the following Table 4.2. Each entry of the Table 4.2 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 4.2. For UMN Plaza1 and Plaza2 the Very-Low depicts to no crowd scenes. The Figure 4.3 shows examples of different congestion levels defined on the following datasets.

Table 4.2: Details of five congestion levels

Datasets	Defining Class Level Densities				
	Very-Low (VL)	**Low (L)**	**Medium (M)**	**High (H)**	**Very-High (VH)**
UCSD_Ped1 [181]	0-8	9-16	17-24	25-32	>32
UCSD_Ped2 [181]	0-8	9-16	17-24	25-32	>32
Pets_2009 [180]	0-8	9-16	17-24	24-32	>32
UMN_Plaza1 [182]	0	1-3	4-6	7-10	>10
UMN_Plaza2 [182]	0	1-3	4-6	7-10	>10

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 4.3 and 4.4 show the division of datasets into "Dataset-1" and "Dataset-2".

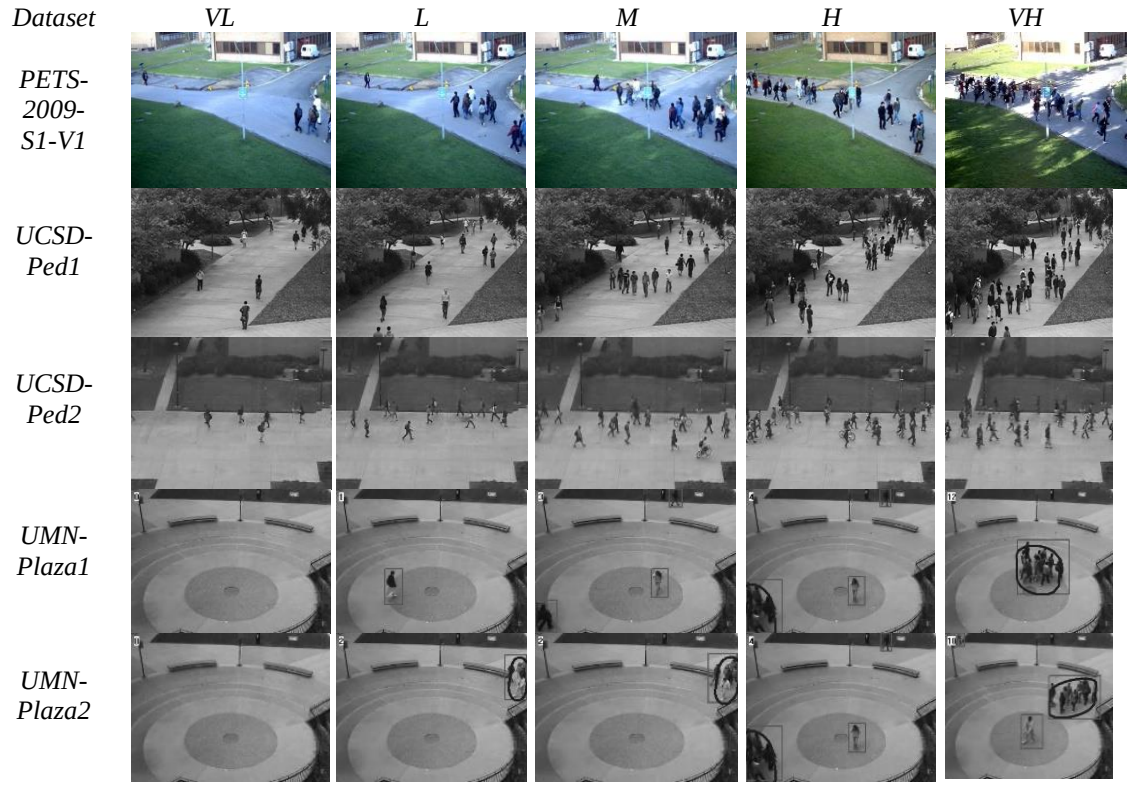

Figure 4.3: Examples of crowd scenes of different crowd congestion-levels

Table 4.3: Details of "Dataset-1"

Dataset Name	Training Samples						Testing Samples					
	VL	L	M	H	VH	Total	VL	L	M	H	VH	Total
UCSD-Ped1 [181]	1713	3808	1462	1006	409	8398	1143	2539	975	671	274	5602
UCSD-Ped2[181]	221	1326	669	322	195	2733	148	885	447	216	131	1827
UMN Plaza1 [182]	116	275	132	65	234	822	78	184	89	44	157	552
UMN Plaza2[182]	91	373	231	234	71	1000	62	249	154	156	48	669
Pets_2009 [180]	52	116	165	102	301	736	35	78	110	68	202	493

Table 4.4: Details of "Dataset-2"

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

4.4 Experimental Setup

The codes for the proposed model were written from scratch in python using Keras libraries and TensorFlow. 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 optimiser [170], the decay rates of first moment i.e., β_1 and second moment i.e., β_2 were set to 0.9 and 0.999 respectively. The batch sizes of 64,128, 256, 64 used for Pets-2009, UCSD-Ped1, UCSD-Ped2, UMN,

respectively. The codes were executed on the laptop containing Intel CORE i7 processor with 8 GB of RAM and 4 GB of NVIDIA GPU.

4.5 Result Analysis and Discussion

The performance analysis is performed by implementing the proposed TIS-MCMS-CNN as well as three state-of-the-art techniques proposed by Fu *et al.* [115], Alzalani *et al.* [114] and Kim *et al*. [123] on the prepared datasets like Dataset-1 and Dataset-2. The Figure 4.4 to Figure 4.13 shows the confusion matrix heatmaps of the proposed model for the Dataset-1 and Dataset-2.

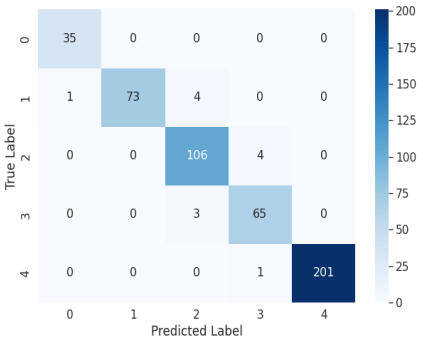

Figure 4.4: Confusion Matrix-Heatmap of TIS-MCMS-CNN for Pets-2009 of Dataset-1.

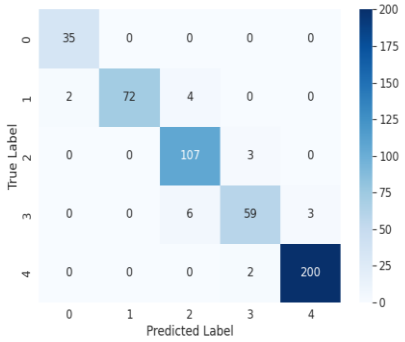

Figure 4.5: Confusion Matrix-Heatmap of TIS-MCMS-CNN for Pets-2009 of Dataset-2.

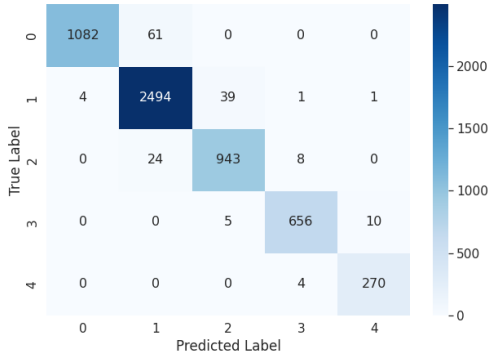

Figure 4.6: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UCSD-Ped1 of Dataset-1.

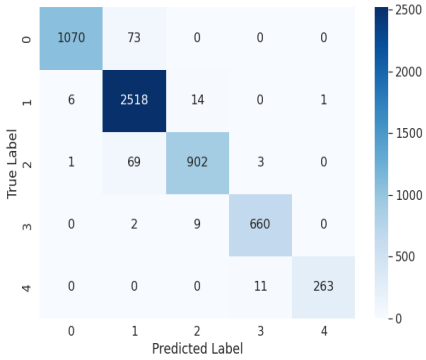

Figure 4.7: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UCSD-Ped1 of Dataset-2.

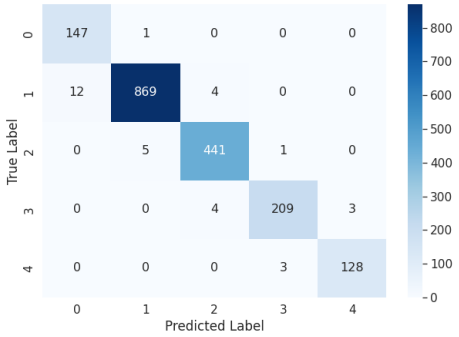

Figure 4.8: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UCSD-Ped2 of Dataset-1.

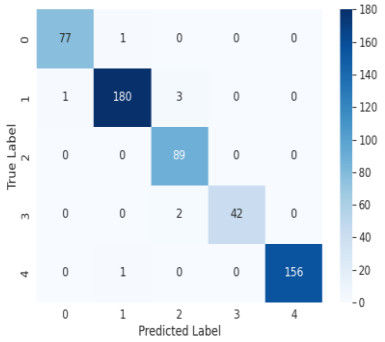

Figure 4.9: Confusion Matrix-Heatmap of TIS-MCMS-CNN of UCSD-Ped2 of Dataset-2.

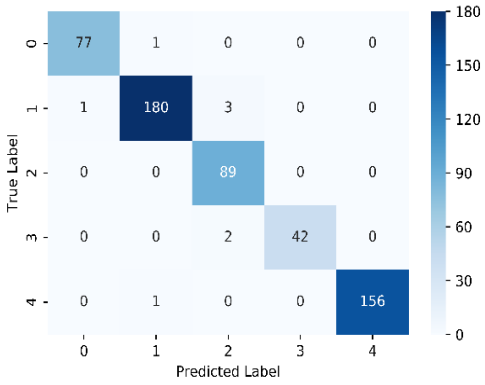

Figure 4.10: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza1 of Dataset-1.

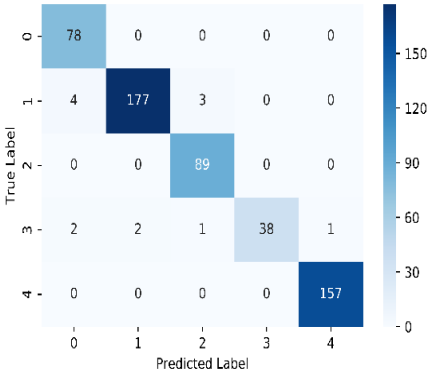

Figure 4.11: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza2 of Dataset-2.

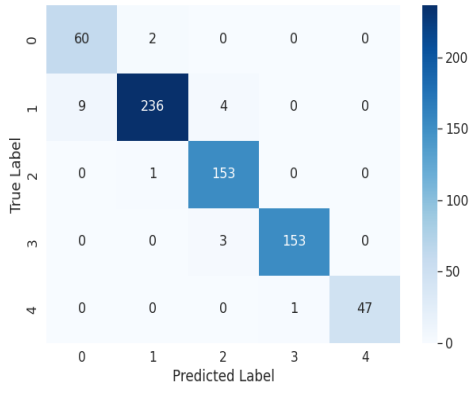

Figure 4.12: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza2 of Dataset-1.

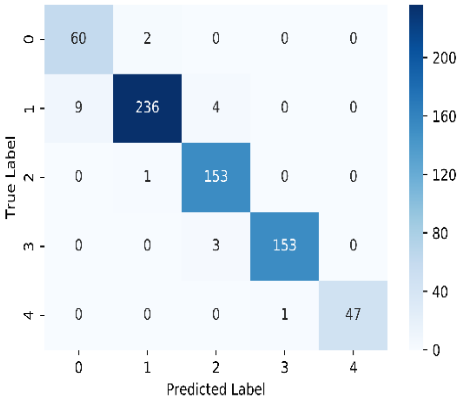

Figure 4.13: Confusion Matrix-Heatmap of TIS-MCMS-CNN for UMN-Plaza2 of Dataset-2.

4.5.1 Pets-2009

Table 4.5 shows the results of different approaches using Dataset-1 and Dataset-2 of Pets-2009. From the Table 4.5, it can be noticed that the proposed architecture provides better results as compared with the state-of-the-art techniques. The values mentioned in bold letters are the best in Table 4.5. 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 the proposed architecture can extract efficient features irrespective of challenging situations and outperforms the MS-CNN [115], CLBP [114], and MLP [123] in terms of Accuracy, Error Rate, Recall (R), Specificity (S), Precision (P), False Positive Rate (FPR) and Fi_Score.

Table 4.5: 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 [115]	0.9412	0.0588	0.9338	0.9856	0.9298	0.0144	0.9309
	CLBP[114]	0.9290	0.0710	0.9104	0.9821	0.9227	0.0179	0.9150
	MLP[123]	0.5071	0.4929	0.4082	0.8635	0.3200	0.1365	0.2920
	TIS- MCMS-CNN	**0.9697**	**0.0303**	**0.9667**	**0.9926**	**0.9624**	**0.0074**	**0.9644**
Dataset-2	MS-CNN [115]	0.9329	0.0671	0.9104	0.9826	0.9285	0.0174	0.9183
	CLBP[114]	0.9290	0.071	0.9085	0.9819	0.9242	0.0181	0.9145
	MLP[123]	0.5010	0.4990	0.3290	0.8580	0.2665	0.1420	0.2896
	TIS- MCMS-CNN	**0.9594**	**0.0406**	**0.9507**	**0.9895**	**0.9535**	**0.0105**	**0.9513**

4.5.2 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 (R), Specificity (S), Precision (P), False positive rate (FPR), and F1-score. Table 4.6 shows details of the achieved performance measurements for different approaches. The values mentioned in bold letters are the best in the table.

Table 4.6: Performance analysis of several approaches on UCSD-Ped1

Dataset	Approaches	Performance Metrics						
		Accuracy	**Error**	**R**	**S**	**P**	**FPR**	**F1_score**
Dataset-1	**MS-CNN [115]**	0.9509	0.0491	0.9536	0.9849	0.9348	0.0151	0.9411
	CLBP[114]	0.9135	0.0865	0.8925	0.9737	0.9224	0.0263	0.9062
	MLP[123]	0.4732	0.5268	0.2654	0.8220	0.3129	0.1780	0.2480
	TIS- MCMS-CNN	**0.9721**	**0.0279**	**0.9719**	**0.9915**	**0.9727**	**0.0085**	**0.9721**
Dataset-2	**MS-CNN [115]**	0.9309	0.0691	0.9269	0.9783	0.9459	0.0217	0.9359
	CLBP[114]	0.9135	0.0865	0.8925	0.9737	0.9224	0.0263	0.9062
	MLP[123]	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**

4.5.3 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 4.7, and it can be noticed that the proposed architecture performs better than other techniques in terms of Error, Recall (R), Specificity (S), Precision (P), False positive rate (FPR), and F1-scores. Thus, it capably handles the challenging situations that exist in the dataset.

Table 4.7: Performance analysis of several approaches on UCSD-Ped2

Dataset	Approaches	Performance Metrics						
		Accuracy	**Error**	**R**	**S**	**P**	**FPR**	**F1_score**
Dataset-1	**MS-CNN [115]**	0.9217	0.0783	0.8943	0.9756	0.9231	0.0244	0.9075
	CLBP[114]	0.8955	0.1045	0.8467	0.9669	0.9040	0.0331	0.8703
	MLP[123]	0.5095	0.4905	0.2560	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 [115]**	0.8747	0.1253	0.8349	0.9633	0.8789	0.0367	0.8511
	CLBP[114]	0.9020	0.0980	0.8608	0.9684	0.9138	0.0316	0.8830
	MLP[123]	0.5014	0.4986	0.2149	0.812	0.2043	0.1880	0.1610
	TIS- MCMS-CNN	**0.9852**	**0.0148**	**0.9766**	**0.9954**	**0.9839**	**0.0046**	**0.9801**

4.5.4 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 Table 4.8 and Table 4.9. The values mentioned in bold letters are the best in the table. Although CLBP [110] performs better in Dataset-2, it takes much time to process a frame. The time analysis for all the approaches is discussed in Section 4.5.6.

Table 4.8: Performance analysis of several approaches on UMN-Plaza1

Dataset	Approaches	Performance Metrics						
		Accuracy	**Error**	**R**	**S**	**P**	**FPR**	**F1_score**
Dataset-1	MS-CNN [115]	0.9511	0.0489	0.9521	0.9880	0.9360	0.0120	0.9435
	CLBP[114]	0.9837	0.0163	0.9856	0.9960	0.9813	0.0040	0.9832
	MLP[123]	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 [115]	0.9511	0.0489	0.9521	0.9880	0.9360	0.0120	0.9435
	CLBP[114]	0.9837	0.0163	0.9856	0.9960	0.9813	0.0040	0.9832
	MLP[123]	0.7100	0.2900	0.6131	0.9232	0.5722	0.0768	0.5880
	TIS- MCMS-CNN	**0.9764**	**0.0236**	**0.9651**	**0.9941**	**0.9736**	**0.0059**	**0.9680**

Table 4.9: Performance analysis of several approaches on UMN-Plaza2

Dataset	Approaches	Performance Metrics						
		Accuracy	**Error**	**R**	**S**	**P**	**FPR**	**F1_score**
Dataset-1	MS-CNN [115]	0.8879	0.1121	0.8884	0.9666	0.9226	0.0334	0.8975
	CLBP[114]	0.7653	0.2347	0.8715	0.9391	0.8876	0.0609	0.8370
	MLP[123]	0.7280	0.2720	0.5770	0.9225	0.6016	0.0774	0.5600
	TIS- MCMS-CNN	**0.9701**	**0.0299**	**0.9738**	**0.9925**	**0.9614**	**0.0075**	**0.9669**
Dataset-2	MS-CNN [115]	0.9178	0.0822	0.9245	0.9749	0.9458	0.0251	0.9314
	CLBP[114]	0.7638	0.2362	0.8702	0.9387	0.8873	0.0613	0.8361
	MLP[123]	0.7250	0.2750	0.5846	0.9218	0.5910	0.0782	0.5681
	TIS- MCMS-CNN	**0.9701**	**0.0299**	**0.9738**	**0.9925**	**0.9614**	**0.0075**	**0.9669**

4.5.5 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 4.10. The values mentioned in bold letters are the best in the table. The accuracies of two streams are very low as

compared with the proposed architecture. It can be concluded from Table 4.10 that neither spatial stream nor the 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.

Table 4.10: Performance analysis of Ablation Study on Different Datasets

Datasets		**Approaches**	**Performance Metrics**						
			Acc	**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**	**0.8960**	**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	0.9950	0.9753	0.0050	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	0.8670	0.1330	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**

4.5.6 Time Analysis

The average time required to process the test frames were also calculated. It was found that the proposed approach achieved around 29.45 (≈30) frames per second. Although MS-MCNN performs better still, the proposed model can provide results in real-time. Table 4.11 shows average frames per second achieved for test cases by different approaches.

Table 4.11: Test frames processing time of several approaches

Frame-rate Vs Approaches	**Approaches**				
	MS-CNN[115]	**MLP**[40]	**CLBP**[123]	**SIS-MCMS-CNN**	**TIS-MCMS-CNN**
Average Execution Time (frames/seconds)	65	25	0.143	43.49	29.45

4.6 Conclusion

In this work, a TIS-MCMS-CNN was proposed to perform crowd congestion-level analysis at the global or frame level. This model extracted features invariant to perspective change and scene changes. The CCA dataset was created using publicly available crowd datasets: Pets-2009, UCSD Ped1, UCSD Ped2, UMN Plaza-1, and UMN Plaza2 by annotating different crowd congestion levels. The created dataset was divided

into two sets named "Dataset-1" and "Dataset-2" based on training, validation, and testing samples. The proposed model achieved accuracies of 96.97%, 97.21%, 98.19%, 97.01% and 98.55% on Dataset-1. For Dataset-2 the proposed model achieved accuracies of 95.94%, 96.63%, 98.52%, 97.01% and 97.64%. In addition, several state-of-the-art approaches were also implemented and experimented on the created dataset. The proposed model outperforms the state-of-the-art techniques in terms of Accuracy (Acc), F1_score, Precision (P), Recall (R), and Specificity (S). The architecture processed an average of nearly 30 test frames per second in real-time.

In this chapter, an efficient model for the CCA is proposed, and discussed its working. An extensive experimental analysis was conducted, which showed the proposed model's superiority over the state-of-the-art models. In the next chapter, another essential task of CA, i.e., CBA using deep learning approaches, will be discussed.