Real-time Traffic Congestion Avoidance for High-Density Urban Intersections using Deep Learning

By

Mohan Ram R (20BCE1742)

A PROJECT REPORT submitted to

Dr. Vedhapriyavadhana R

Associate Professor, School of Computer Science and Engineering

in partial fulfillment of the requirements for the course of

CSE4019 - Image Processing

in

B. Tech. Computer Science and Engineering School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Abstract

This study presents a state-of-the-art method for high-density urban intersections traffic congestion avoidance in real time. This method transforms traffic management by utilising the strength of deep learning applied to photos. It makes use of sophisticated computer vision algorithms to forecast congestion hotspots and analyse traffic dynamics. Instantaneous detection allows the system to modify traffic signals, redirect vehicles, and optimise traffic flow in real-time, reducing traffic and improving urban mobility. In highly populated urban regions, this novel strategy offers a viable path towards effective and sustainable traffic management, with significant advantages in terms of lowering travel times and fuel consumption.

Introduction

Urban traffic management has advanced to a new level using Deep Learning-powered Real-time Traffic Congestion Avoidance for High-Density Urban Intersections. High-density junctions are a recurring problem in today's busy cities, affecting the quality of life for commuters, economic output, and environmental sustainability. This creative idea has the potential to completely transform how we navigate these intricate metropolitan settings by utilising Deep Learning.

Neural network designs developed by Deep Learning are highly skilled at extracting complex patterns from large amounts of data. These models become effective instruments for real-time congestion prediction, analysis, and mitigation when used in traffic management.

The incorporation of neural networks trained on a variety of data sources, including as real-time feeds, sensors, traffic cameras, and historical traffic patterns, is essential to this approach. Through Convolutional Neural Networks (CNNs), the system analyses live streams from intersections, extracting spatial and temporal features to identify traffic flow, vehicle density, and anomalies in real time. This technology promises reduced commute times, minimized environmental impact, and optimized resource allocation for transportation infrastructure, aiming to transform urban intersections into efficient, interconnected nodes within the urban landscape.

Related Work

1. **Real-time image enhancement for an automatic automobile accident detection through CCTV using deep learning**

The balance between processing resources and detection accuracy is a prevalent problem in automatic accident detection (AAD) systems, and it is addressed in this study. Three stages—Detection, Tracking, and Classification—each with lower computational demands—are proposed by the authors as a framework for organising the AAD system. They introduce Mini-YOLO, a deep learning model for detection that has accuracy comparable to YOLO but is smaller and has less processing cost. On a low-end PC, Mini-YOLO manages an astounding 28 frames per second. For classification, a support vector machine with a radial basis kernel proves to be the most effective in terms of memory and latency. The tracking step uses SORT (Simple Online Real-time Tracking). Overall, this paper presents a practical

and computationally efficient AAD system suitable for real-time deployment, with a focus on Mini-YOLO's exceptional runtime speed and the efficiency of the chosen classification algorithm.

2. **Freeway accident detection and classification based on the multi-vehicle trajectory data and deep learning model**

This paper examines the identification and classification of motorway accidents, a topic of great interest to academics. There is a wealth of real-time vehicle trajectory data available due to the increasing usage of Global Navigation Satellite Systems (GNSS) in mobile devices and onboard equipment, which presents an alluring avenue for accident detection and categorization. The suggested approach is unique in that it tracks the evolution of accident vehicle trajectory patterns over time in addition to GNSS positioning data. To distinguishing accidents from routine vehicular activity and identifying the accident type, a Deep Convolutional Neural Network (DCNN) model with six categories is introduced. The study emphasises the necessity of trajectory data with a minimum duration and frequency for reliable detection accuracy. While some accident types, like lane-change incidents, are simpler to spot, others, like rear-end accidents, present greater difficulties. The study is expanded to include curving roads, accuracy for accident kinds is improved, and more accident categories are taken into consideration in the paper's suggestions for future research areas.

3. **Sensing Accident-Prone Features in Urban Scenes for Proactive Driving and Accident Prevention**

To address potential driver distractions brought on by visual information in metropolitan environments that can cause them to miss traffic signs and accident-prone (AP) features, this research presents a novel method. The authors suggest a real-time visual notification system for drivers based on dashcam photos to lessen the danger of missing these signs. To train an attention module that will categorise urban settings as either accident hotspots or non-hotspots, they use Google Street View photographs from accident hotspots that were determined from actual accident data. This attention module integrates channel, point, and spatial-wise attention learning and improves the performance of CNN backbones. Up to 92% classification accuracy is achieved, which is astounding. The study supports the creation of a global dataset for enhanced generalisation by demonstrating the model's flexibility across various urban cities.

4. **Deep Learning on Traffic Prediction: Methods, Analysis, and Future Directions**

To effectively manage traffic congestion, route planning, vehicle dispatching, and other aspects of intelligent transportation systems are highlighted in this study. The deep and dynamic spatio-temporal linkages that exist within road networks among various locations are what give rise to the complexity. A thorough analysis of deep learning-based methods in this field is required considering recent improvements in deep learning's capacity for traffic prediction. By first summarising existing traffic prediction techniques and offering a taxonomy for clarity, the study presents a multifaceted approach. The most recent methodologies for diverse traffic prediction applications are then presented, and a collection of commonly used public datasets is compiled to assist other researchers.

5. **A novel image-based convolutional neural network approach for traffic congestion estimation**

This study addresses the drawbacks of conventional image-based traffic congestion estimating techniques, which comprise the extraction of vehicles from surveillance photos and the subsequent calculation of the congestion index. These methods take a long time and are not appropriate for detecting traffic congestion in real time. They present a system for estimating traffic congestion based on images that incorporates a convolutional neural network (CNN) model with a traffic parameter layer. This innovative framework makes it possible to calculate and estimate traffic congestion directly, which speeds up processing and does not require complicated post-processing. They use a dataset with 66,890 vehicles in 1,400 traffic photos to train their CNN model, then they use another dataset with 113,516 vehicles in 2,400 traffic images to test it.

6. **A Vehicle Detection Technique Using Binary Images for Heterogeneous and Lane-Less Traffic**

This paper emphasises the need of lane less traffic situations and low-cost traffic control systems for emerging nations. Virtual loops in video and micro-LiDAR arrays are used to show the method and to gather data on vehicle width, length, and height to enable vehicle classification. In comparison to traditional image or video processing techniques, this technology offers benefits including minimal storage requirements, cost-effectiveness, low bandwidth requirements, and less computational complexity. By altering observation zone lengths, it can respond to different traffic situations and achieve detection accuracy of 98% with video data and 91.3% with micro-LiDARs.

7. **Deep traffic congestion prediction model based on road segment grouping**

The approach for creating a traffic congestion index (TCI) by mining free-stream speed and flow data is presented in this study. To maximise deep learning model training, it uses a road segment grouping technique based on association subgraphs, enabling information sharing between road segments. The resulting SG-CNN traffic congestion prediction model performs more accurately than other methods. By including more variables, such as the weather and road conditions, and by creating more effective segment grouping algorithms, future research seeks to improve TCI accuracy and perhaps apply this strategy to other technical problems.

8. **Deep Learning and Remote Sensing: Detection of Dumping Waste Using UAV**

Using UAV photos in Senegal, West Africa, this research proposes an automated method for the detection of covert garbage dumps. These photos' exceptional spatial resolution necessitates the use of sophisticated analytical techniques. The suggested method entails image segmentation, scaling for CNN input, labelling regions of interest, and feature extraction using Single Shot Detector (SSD). The model successfully locates significant places; however, it has trouble in areas where there are no precise ground truth data. With locally gathered data, deep learning and remote sensing have the greatest potential to address a variety of environmental concerns while also providing specialised monitoring and

planning tools. Future potential includes the creation of a thorough supermodel capable of solving numerous issues on its own.

9. Traffic Prediction in Smart Cities Based on Hybrid Feature Space

To predict traffic congestion, this work offers a hybrid GRU-LSTM deep learning model that makes use of city-wide data from several sources. For tasks like map matching, sparsity managing, and outlier elimination, the authors created a data pipeline. Their method outperforms existing deep neural network models and reaches a remarkable 95% accuracy. The approach incorporates traffic volume, GPS, weather, exceptional circumstances, and Open Street Map (OSM) data into a hybrid feature space. When it comes to reducing traffic congestion on arterial highways, deep learning techniques, in particular GRU-LSTM, outperform conventional approaches.

10. Monitoring and surveillance of urban road traffic using low altitude drone images: a deep learning approach

Using cutting-edge Deep Learning (DL) object detection models on the AU-AIR dataset, this research proposes novel algorithms for monitoring aerial picture traffic. The most effective model, YOLOv4, outperforms the competition by at least 88% in mAP. It delivers high real-time applicability and a detection speed that is nearly six times faster. The suggested approach is a competitive option for automatic traffic monitoring and surveillance, but there is still room for advancements like faster inferences and more solid datasets. Alternative methods, including oriented bounding boxes, 3D-CNN, and generative adversarial networks (GANs), will be investigated in future studies.

11. A survey of deep learning techniques for vehicle detection from UAV images

A review of deep learning methods for on-ground vehicle detection from UAV photos is presented in this work. These methods can be used in disaster relief operations, parking management, and traffic management. The survey groups the works depending on how they go about reducing computation overhead and improving accuracy, highlighting their commonalities and contrasts. Future difficulties are also discussed, such as the requirement for on-board image processing, portable neural networks, and better data fusion with other sensors. The labelling procedure for training data should be streamlined while researchers are urged to investigate multiple evaluation criteria beyond predicting performance and concentrate on high-quality, diversified datasets.

12. A hybrid deep learning based intrusion detection system using spatial-temporal representation of in-vehicle network traffic

To improve automobile cybersecurity, this work introduces a hybrid deep learning-based intrusion detection system (HyDL-IDS). To automatically extract geographical and temporal information from in-vehicle network traffic data, the system integrates convolutional neural network (CNN) and long short-term memory (LSTM). Impressive results were obtained when testing against a benchmark dataset for automotive hacking, detecting almost all types

of cyberattacks, including denial-of-service (DoS), fuzzy assaults, and spoofing (Gear and RPM) attacks, while exhibiting a low false alarm rate.

13. Anomaly Detection in Traffic Surveillance Videos Using Deep Learning

Automatic abnormal activity identification is becoming necessary due to the growing prevalence of surveillance cameras, especially in traffic recordings where accidents are to be addressed. This study focuses on using convolutional neural networks (CNNs) in video traffic surveillance systems (VTSS) to detect accidents. With the aid of data augmentation techniques, a specialised dataset known as the Vehicle Accident Image Dataset (VAID) was created for CNN training. When the trained CNN model was tested on several recordings, it was able to identify traffic incidents with an accuracy of 82%. For difficult jobs, deep learning, in particular CNNs, is successful. However, controlling label flickering during testing presents problems, which were resolved by employing a rolling prediction average technique.

14. Object Interaction-Based Localization and Description of Road Accident Events Using Deep Learning

In order to effectively detect accidents in real-time, this paper proposes a novel technique that makes use of object positions. By extracting object interactions and categorising normal and accident interactions iteratively, they divide films into pre-accident, accident, and post-accident stages. High-level textual descriptions quantify context and severity while heat maps draw attention to the accident site. Extensive tests on the UCF Crime and CADP datasets reveal cutting-edge performance. Future developments might consider dynamic camera viewpoints, temporal data, and reliable object detection in various environments.

15. Decentralized Autonomous Navigation of a UAV Network for Road Traffic Monitoring

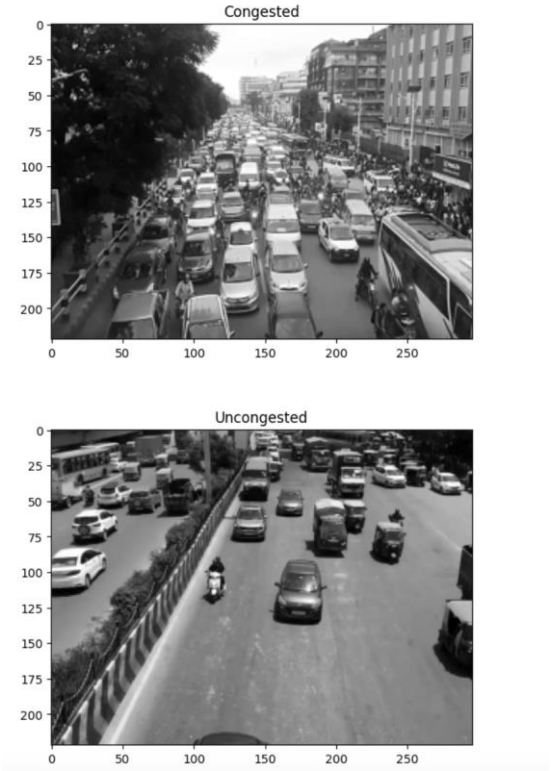
This article examines a different strategy for effective aerial surveillance utilising unmanned aerial vehicles (UAVs). The use of a UAV network to track traffic is the main goal. A decentralised navigation system is suggested, enabling UAVs to identify obstructions and congregate over them to improve ground vehicle visibility. This system, which can be implemented in real-time, depends on light measurements and location sharing among UAVs. Simulations show how effective it is.

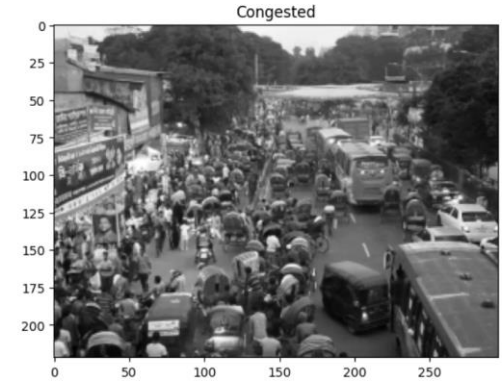
Methodology

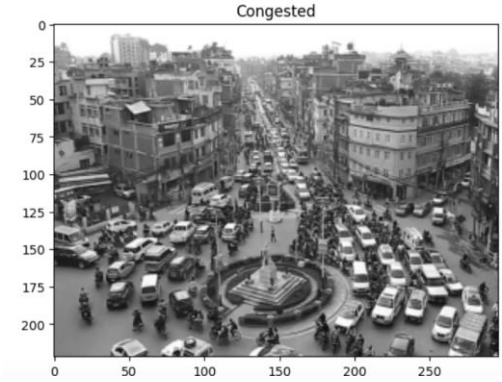
A Convolutional Neural Network (CNN), a specialised architecture skilled in extracting complex patterns and characteristics from picture data, is used by the model to function. In order to restore the original visual structure, the model first processes the dataset by flattening the pixel arrays and reconstructing them into 296 by 222 matrices. Now that they are properly organised, these images go through a normalisation process that uniformizes pixel values for efficient neural network processing. The dataset is divided into separate subsets for testing, validation, and training in order to maintain the relationship between labels and images in each subset.

The CNN layers of the model provide the foundation for its operation. Convolutional layers are used in image processing to identify different visual characteristics by applying many filters to small, overlapping regions of the image to learn patterns. As the network gets deeper, these filters eventually learn more complicated representations, extracting properties like edges, textures, and forms. The ensuing pooling layers simplify computing while condensing and preserving important information. The model is then able to generate predictions based on the attributes it has learned by passing the learned representations through fully linked (dense) layers. These trained representations and the correlations between extracted characteristics provide the model its prediction power, which in turn allows it to distinguish between congested and uncongested states based on patterns found in the photos.

Dataset

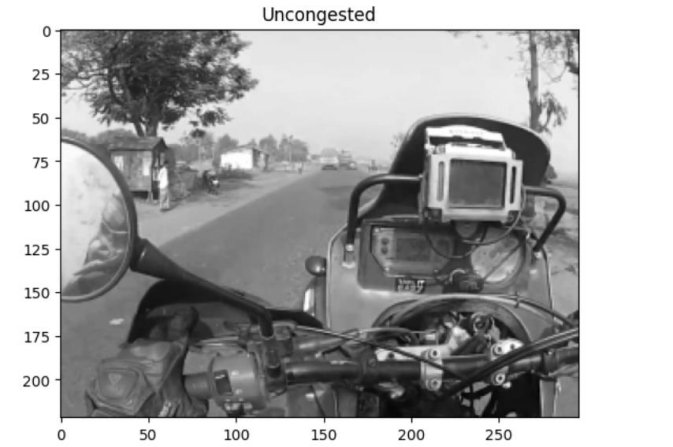



Congested
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250


Congested
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250


Uncongested
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250


Uncongested
0
25
50
75
100
125
150
175
200
0
50
100
150
200
250

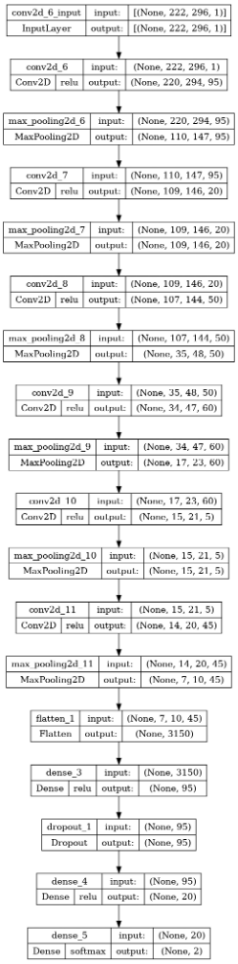


The dataset consists of 5750 image data rows, with each image having a resolution of 296 by 222 pixels. Each row in the dataset has 65713 columns, where the first column represents the labelling feature. The labelling feature is binary, with a value of 1 indicating congestion and a value of 0 representing uncongested conditions.

The remaining 65712 columns correspond to the pixel-values of the associated image. Each pixel has a single pixel-value associated with it, ranging from 0 to 255. The pixel-values represent the lightness or darkness of the pixel, with higher values indicating darker shades.

To locate a specific pixel in the image, you can decompose the column index using the formula x = i * 222 + j, where i and j are integers ranging from 0 to 295 and 0 to 221, respectively. This will help you determine the position of a particular pixel within the 296 by 222 image matrix.

CNN Architecture



Implementation

Importing and Transforming Dataset

In [3]:
```
def label_counter(inpt, data_name):
    c = 0
    u = 0
    for i in inpt:
        if i == 1:
            c = c+1
        elif i == 0:
            u = u + 1
    print(f'{data_name} --> Congested: {c}, Uncongested: {u}')
```

In [4]:
```
def import_dataset(split_seed):

    #First dataset
    congested = pl.read_csv('/kaggle/input/traffic/model2_congested_traffic_dataset.csv/model2_congested_traffic_dataset.csv')

    #Second dtaset
    uncongested = pl.read_csv('/kaggle/input/traffic/model2_uncongested_traffic_dataset.csv/model2_uncongested_traffic_dataset.csv')

    #Combining first and second dataset
    df = pl.concat([congested, uncongested], how="vertical")

    y = df[:, 0] # Getting labels as series
    y = y.to_numpy()

    df = df[:, 1:]/255.0 # Normalizing pixels value to range of 0 to 1
    df = df.to_numpy().reshape(-1, 222, 296, 1) #reshaping to 296 by 222

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.1, random_state=split_seed) #42

    # Split train data into train and validation sets
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15, random_state=split_seed)

    label_counter(y_train, 'y_train')
    label_counter(y_test, 'y_test')
    label_counter(y_val, 'y_val')

    return X_train, y_train, X_test, y_test, X_val, y_val
```

In [5]:
```
X_train, y_train, X_test, y_test, X_val, y_val = import_dataset(25)
```

```
y_train --> Congested: 1878, Uncongested: 1862
y_test --> Congested: 255, Uncongested: 235
y_val --> Congested: 328, Uncongested: 333
```

Exploratory Data Analysis

In [6]:
```
import plotly.graph_objects as go
# Define class labels
class_labels = {0: 'Congested images', 1: 'Uncongested images'}
```

In [7]:
```
# Map class labels for visualization
train_classes = np.array([class_labels[label] for label in y_train])
test_classes = np.array([class_labels[label] for label in y_test])
val_classes = np.array([class_labels[label] for label in y_val])
```

In [8]:
```
# Calculate class frequencies
train_class_counts = np.unique(train_classes, return_counts=True)
test_class_counts = np.unique(test_classes, return_counts=True)
val_class_counts = np.unique(val_classes, return_counts=True)
```

In [9]:
```
# Create bar chart data
data = [
    go.Bar(x=train_class_counts[0], y=train_class_counts[1], name='Train'),
    go.Bar(x=test_class_counts[0], y=test_class_counts[1], name='Test'),
    go.Bar(x=val_class_counts[0], y=val_class_counts[1], name='Validation')
]
```

In [10]:
```
# Set layout
layout = go.Layout(
    title='Occurrences of Binary Classes',
    xaxis=dict(title='Class'),
    yaxis=dict(title='Count'),
    barmode='group'
)

# Create figure
fig = go.Figure(data=data, layout=layout)

# Show the bar chart
fig.show()
```

In [11]:

```
# Define class labels
class_labels = {
    0: 'Congested',
    1: 'Uncongested'
}

# Calculate total class counts
class_counts = {
    class_labels[0]: np.sum([np.sum(y_train == 0), np.sum(y_test == 0), np.sum(y_val == 0)]),
    class_labels[1]: np.sum([np.sum(y_train == 1), np.sum(y_test == 1), np.sum(y_val == 1)])
}

# Create bar chart data
data = [
    go.Bar(x=list(class_counts.keys()), y=list(class_counts.values()))
]

# Set layout
layout = go.Layout(
    title='Total Occurrences of Binary Classes',
    xaxis=dict(title='Class'),
    yaxis=dict(title='Count')
)

# Create figure
fig = go.Figure(data=data, layout=layout)

# Show the bar chart
fig.show()
```

In [12]:

```
def image_visualizer(X, y, row_number):
    X = X[row_number]
    y = y[row_number]

    plt.imshow(X, cmap='gray')
    lbl = None
    if y == 0:
        lbl = 'Uncongested'
    elif y == 1:
        lbl = 'Congested'
    plt.title(lbl)
    plt.show()

image_visualizer(X_train, y_train, 5)
image_visualizer(X_train, y_train, 225)
image_visualizer(X_train, y_train, 512)
image_visualizer(X_train, y_train, 995)
image_visualizer(X_train, y_train, 15)
image_visualizer(X_train, y_train, 1225)
image_visualizer(X_train, y_train, 1512)
image_visualizer(X_train, y_train, 1995)
image_visualizer(X_train, y_train, 25)
image_visualizer(X_train, y_train, 2225)
image_visualizer(X_train, y_train, 2512)
image_visualizer(X_train, y_train, 2995)
```

Model

In [13]:
```
def model_builder(hp):
    model = keras.Sequential()

    model.add(keras.layers.Conv2D(hp.Int('filter_1', min_value=5, max_value=100, step=10), hp.Choice
('filter_size_1', values=[3, 4, 5]), activation='relu', input_shape=(222, 296, 1)))
    model.add(keras.layers.MaxPooling2D(hp.Choice('kernel_size_1', values=[1, 2, 3, 4])))

    model.add(keras.layers.Conv2D(hp.Int('filter_2', min_value=5, max_value=70, step=5), hp.Choice
('filter_size_2', values=[2, 3]), activation='relu'))
    model.add(keras.layers.MaxPooling2D(hp.Choice('kernel_size_2', values=[1, 2, 3, 4])))

    model.add(keras.layers.Conv2D(hp.Int('filter_3', min_value=5, max_value=70, step=5), hp.Choice
('filter_size_3', values=[2, 3]), activation='relu'))
    model.add(keras.layers.MaxPooling2D(hp.Choice('kernel_size_3', values=[1, 2, 3, 4])))

    model.add(keras.layers.Conv2D(hp.Int('filter_4', min_value=5, max_value=70, step=5), hp.Choice
('filter_size_4', values=[2, 3]), activation='relu'))
    model.add(keras.layers.MaxPooling2D(hp.Choice('kernel_size_4', values=[1, 2, 3])))

    model.add(keras.layers.Conv2D(hp.Int('filter_5', min_value=5, max_value=70, step=5), hp.Choice
('filter_size_5', values=[1, 2, 3]), activation='relu'))
    model.add(keras.layers.MaxPooling2D(hp.Choice('kernel_size_5', values=[1, 2, 3])))

    model.add(keras.layers.Conv2D(hp.Int('filter_6', min_value=5, max_value=80, step=5), hp.Choice
('filter_size_6', values=[1, 2, 3]), activation='relu'))
    model.add(keras.layers.MaxPooling2D(hp.Choice('kernel_size_6', values=[1, 2, 3])))

    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(units=hp.Int('units_1', min_value=5, max_value=100, step=10), activ
ation='relu'))

    hp_dropout_rate = hp.Float('dropout_rate', min_value=0.1, max_value=0.5, step=0.1)
    model.add(keras.layers.Dropout(hp_dropout_rate))

    model.add(keras.layers.Dense(hp.Int('units_2', min_value=5, max_value=55, step=5), activation='r
elu'))

    model.add(keras.layers.Dense(2, activation='softmax'))

    hp_learning_rate = hp.Choice('learning_rate', values=[0.01, 0.001, 0.0003, 0.0001])

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                  loss=keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])

    return model
```

In [15]:
```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='/kaggle/working/',
                     project_name='congested_vs_uncongested_traffic_detector')

stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)

tuner.search(X_train, y_train, epochs=15, validation_data=(X_val, y_val), callbacks=[stop_early])
```

```
Trial 23 Complete [00h 01m 14s]
val_accuracy: 0.8078668713569641

Best val_accuracy So Far: 0.9788199663162231
Total elapsed time: 00h 13m 01s
```

In [16]:
```
# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]
```

In [17]:
```
model = tuner.hypermodel.build(best_hps)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 220, 294, 95)      950

max_pooling2d_6 (MaxPooling  (None, 110, 147, 95)     0
2D)

conv2d_7 (Conv2D)            (None, 109, 146, 20)      7620

max_pooling2d_7 (MaxPooling  (None, 109, 146, 20)     0
2D)

conv2d_8 (Conv2D)            (None, 107, 144, 50)      9050

max_pooling2d_8 (MaxPooling  (None, 35, 48, 50)       0
2D)

conv2d_9 (Conv2D)            (None, 34, 47, 60)        12060

max_pooling2d_9 (MaxPooling  (None, 17, 23, 60)       0
2D)

conv2d_10 (Conv2D)           (None, 15, 21, 5)         2705

max_pooling2d_10 (MaxPoolin  (None, 15, 21, 5)        0
g2D)

conv2d_11 (Conv2D)           (None, 14, 20, 45)        945

max_pooling2d_11 (MaxPoolin  (None, 7, 10, 45)        0
g2D)

flatten_1 (Flatten)          (None, 3150)              0

dense_3 (Dense)              (None, 95)                299345

dropout_1 (Dropout)          (None, 95)                0

dense_4 (Dense)              (None, 20)                1920

dense_5 (Dense)              (None, 2)                 42

=================================================================
Total params: 334,637
Trainable params: 334,637
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
# Data augmentation
datagen = keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

In [20]:
```
history = model.fit(datagen.flow(X_train, y_train),
                    validation_data=(X_val, y_val),
                    epochs=60)
```

```
Epoch 1/60
117/117 [==============================] - 20s 146ms/step - loss: 0.6559 - accuracy: 0.5930
- val_loss: 0.4130 - val_accuracy: 0.8139
Epoch 2/60
117/117 [==============================] - 17s 140ms/step - loss: 0.3508 - accuracy: 0.8580
- val_loss: 0.4635 - val_accuracy: 0.7685
Epoch 3/60
117/117 [==============================] - 16s 140ms/step - loss: 0.3130 - accuracy: 0.8703
- val_loss: 0.4322 - val_accuracy: 0.8018
Epoch 4/60
117/117 [==============================] - 17s 141ms/step - loss: 0.2731 - accuracy: 0.9000
- val_loss: 0.1929 - val_accuracy: 0.9380
Epoch 5/60
117/117 [==============================] - 17s 140ms/step - loss: 0.2689 - accuracy: 0.8987
- val_loss: 0.2229 - val_accuracy: 0.9213
Epoch 6/60
117/117 [==============================] - 16s 139ms/step - loss: 0.2452 - accuracy: 0.9102
- val_loss: 0.1976 - val_accuracy: 0.9213
Epoch 7/60
117/117 [==============================] - 17s 141ms/step - loss: 0.2254 - accuracy: 0.9128
- val_loss: 0.2007 - val_accuracy: 0.9304
Epoch 8/60
117/117 [==============================] - 16s 138ms/step - loss: 0.2004 - accuracy: 0.9241
- val_loss: 0.1827 - val_accuracy: 0.9349
Epoch 9/60
117/117 [==============================] - 17s 142ms/step - loss: 0.2071 - accuracy: 0.9243
- val_loss: 0.2021 - val_accuracy: 0.9198
Epoch 10/60
117/117 [==============================] - 16s 138ms/step - loss: 0.1916 - accuracy: 0.9289
- val_loss: 0.1563 - val_accuracy: 0.9395
Epoch 11/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1905 - accuracy: 0.9289
- val_loss: 0.1582 - val_accuracy: 0.9501
Epoch 12/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1794 - accuracy: 0.9380
- val_loss: 0.1423 - val_accuracy: 0.9516
Epoch 13/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1776 - accuracy: 0.9334
- val_loss: 0.1189 - val_accuracy: 0.9667
Epoch 14/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1568 - accuracy: 0.9455
- val_loss: 0.1083 - val_accuracy: 0.9697
Epoch 15/60
117/117 [==============================] - 16s 139ms/step - loss: 0.1599 - accuracy: 0.9439
- val_loss: 0.1255 - val_accuracy: 0.9546
Epoch 16/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1650 - accuracy: 0.9398
- val_loss: 0.1289 - val_accuracy: 0.9516
Epoch 17/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1606 - accuracy: 0.9465
- val_loss: 0.1109 - val_accuracy: 0.9652
Epoch 18/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1437 - accuracy: 0.9513
- val_loss: 0.1270 - val_accuracy: 0.9622
Epoch 19/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1496 - accuracy: 0.9471
- val_loss: 0.0904 - val_accuracy: 0.9728
Epoch 20/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1382 - accuracy: 0.9495
- val_loss: 0.1733 - val_accuracy: 0.9380
Epoch 21/60
117/117 [==============================] - 17s 144ms/step - loss: 0.1512 - accuracy: 0.9487
- val_loss: 0.0911 - val_accuracy: 0.9758
Epoch 22/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1450 - accuracy: 0.9463
- val_loss: 0.0978 - val_accuracy: 0.9773
Epoch 23/60
117/117 [==============================] - 17s 141ms/step - loss: 0.1513 - accuracy: 0.9414
- val_loss: 0.1973 - val_accuracy: 0.9153
Epoch 24/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1407 - accuracy: 0.9479
- val_loss: 0.0896 - val_accuracy: 0.9682
```

```
Epoch 25/60
117/117 [==============================] - 17s 140ms/step - loss: 0.1370 - accuracy: 0.9495
- val_loss: 0.1698 - val_accuracy: 0.9334
Epoch 26/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1453 - accuracy: 0.9473
- val_loss: 0.1476 - val_accuracy: 0.9365
Epoch 27/60
117/117 [==============================] - 17s 144ms/step - loss: 0.1304 - accuracy: 0.9575
- val_loss: 0.1233 - val_accuracy: 0.9607
Epoch 28/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1333 - accuracy: 0.9508
- val_loss: 0.1417 - val_accuracy: 0.9486
Epoch 29/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1301 - accuracy: 0.9586
- val_loss: 0.0821 - val_accuracy: 0.9728
Epoch 30/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1380 - accuracy: 0.9519
- val_loss: 0.0906 - val_accuracy: 0.9758
Epoch 31/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1297 - accuracy: 0.9564
- val_loss: 0.1070 - val_accuracy: 0.9652
Epoch 32/60
117/117 [==============================] - 17s 141ms/step - loss: 0.1366 - accuracy: 0.9500
- val_loss: 0.0754 - val_accuracy: 0.9803
Epoch 33/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1221 - accuracy: 0.9553
- val_loss: 0.0905 - val_accuracy: 0.9743
Epoch 34/60
117/117 [==============================] - 16s 139ms/step - loss: 0.1248 - accuracy: 0.9575
- val_loss: 0.0871 - val_accuracy: 0.9758
Epoch 35/60
117/117 [==============================] - 17s 141ms/step - loss: 0.1184 - accuracy: 0.9572
- val_loss: 0.1023 - val_accuracy: 0.9652
Epoch 36/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1213 - accuracy: 0.9586
- val_loss: 0.1165 - val_accuracy: 0.9592
Epoch 37/60
117/117 [==============================] - 17s 141ms/step - loss: 0.1294 - accuracy: 0.9551
- val_loss: 0.0913 - val_accuracy: 0.9682
Epoch 38/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1109 - accuracy: 0.9594
- val_loss: 0.1081 - val_accuracy: 0.9637
Epoch 39/60
117/117 [==============================] - 16s 140ms/step - loss: 0.1198 - accuracy: 0.9556
- val_loss: 0.0803 - val_accuracy: 0.9728
Epoch 40/60
117/117 [==============================] - 17s 141ms/step - loss: 0.1124 - accuracy: 0.9626
- val_loss: 0.0727 - val_accuracy: 0.9758
Epoch 41/60
117/117 [==============================] - 17s 145ms/step - loss: 0.1096 - accuracy: 0.9634
- val_loss: 0.0833 - val_accuracy: 0.9697
Epoch 44/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1118 - accuracy: 0.9631
- val_loss: 0.0935 - val_accuracy: 0.9697
Epoch 45/60
117/117 [==============================] - 17s 144ms/step - loss: 0.1098 - accuracy: 0.9610
- val_loss: 0.0619 - val_accuracy: 0.9788
Epoch 46/60
117/117 [==============================] - 17s 141ms/step - loss: 0.1030 - accuracy: 0.9647
- val_loss: 0.1328 - val_accuracy: 0.9516
Epoch 47/60
117/117 [==============================] - 17s 142ms/step - loss: 0.1150 - accuracy: 0.9612
- val_loss: 0.0764 - val_accuracy: 0.9743
Epoch 48/60
117/117 [==============================] - 17s 143ms/step - loss: 0.1029 - accuracy: 0.9644
- val_loss: 0.1042 - val_accuracy: 0.9652
Epoch 49/60
117/117 [==============================] - 17s 144ms/step - loss: 0.0967 - accuracy: 0.9671
- val_loss: 0.0636 - val_accuracy: 0.9773
Epoch 50/60
117/117 [==============================] - 16s 138ms/step - loss: 0.0982 - accuracy: 0.9658
- val_loss: 0.0819 - val_accuracy: 0.9773
```

```
Epoch 51/60
117/117 [==============================] - 16s 139ms/step - loss: 0.1039 - accuracy: 0.9615 - val
_loss: 0.0777 - val_accuracy: 0.9728
Epoch 52/60
117/117 [==============================] - 17s 143ms/step - loss: 0.0903 - accuracy: 0.9687 - val
_loss: 0.0907 - val_accuracy: 0.9697
Epoch 53/60
117/117 [==============================] - 17s 144ms/step - loss: 0.1122 - accuracy: 0.9599 - val
_loss: 0.0634 - val_accuracy: 0.9818
Epoch 54/60
117/117 [==============================] - 17s 143ms/step - loss: 0.0894 - accuracy: 0.9706 - val
_loss: 0.1066 - val_accuracy: 0.9637
Epoch 55/60
117/117 [==============================] - 17s 145ms/step - loss: 0.0987 - accuracy: 0.9652 - val
_loss: 0.0791 - val_accuracy: 0.9682
Epoch 56/60
117/117 [==============================] - 17s 148ms/step - loss: 0.0949 - accuracy: 0.9655 - val
_loss: 0.0740 - val_accuracy: 0.9728
Epoch 57/60
117/117 [==============================] - 17s 144ms/step - loss: 0.1203 - accuracy: 0.9543 - val
_loss: 0.0729 - val_accuracy: 0.9773
Epoch 58/60
117/117 [==============================] - 18s 149ms/step - loss: 0.0927 - accuracy: 0.9687 - val
_loss: 0.0567 - val_accuracy: 0.9834
Epoch 59/60
117/117 [==============================] - 17s 145ms/step - loss: 0.0945 - accuracy: 0.9679 - val
_loss: 0.0846 - val_accuracy: 0.9682
Epoch 60/60
117/117 [==============================] - 17s 145ms/step - loss: 0.0962 - accuracy: 0.9636 - val
_loss: 0.0826 - val_accuracy: 0.9713
```

In [21]:

```
val_acc_per_epoch = history.history['val_accuracy']
best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print('Best epoch: %d' % (best_epoch,))
```

```
Best epoch: 58
```

In [22]:

```
index = 1
print(y_test[index])
model.predict(X_test[index].reshape(-1, 222, 296, 1))
```

```
1
1/1 [==============================] - 0s 367ms/step
```

Out[22]:

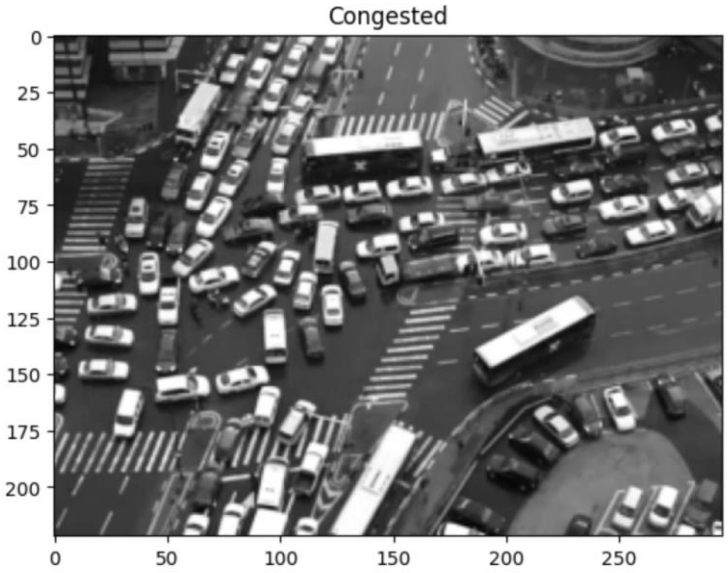
```
array([[0.00125917, 0.99874085]], dtype=float32)
```

In [23]:

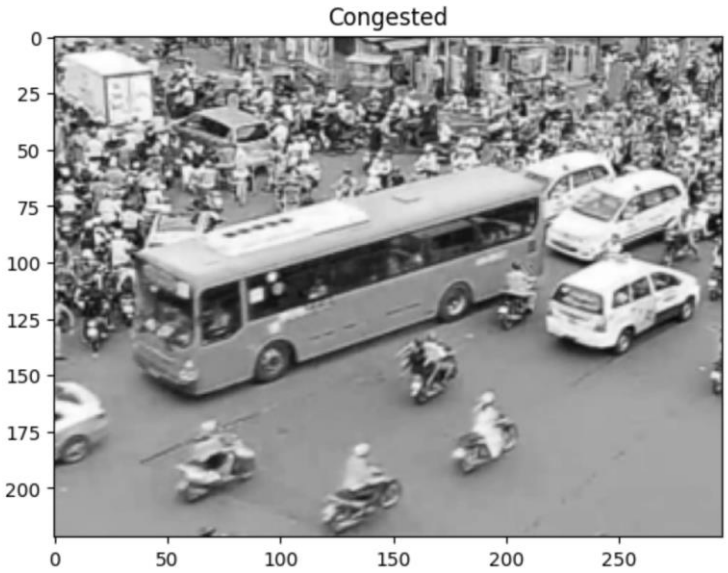
```
model.save('/kaggle/working/model')
```

Results

[5]:
```
image_visualizer(X_val, y_val, 7)
```

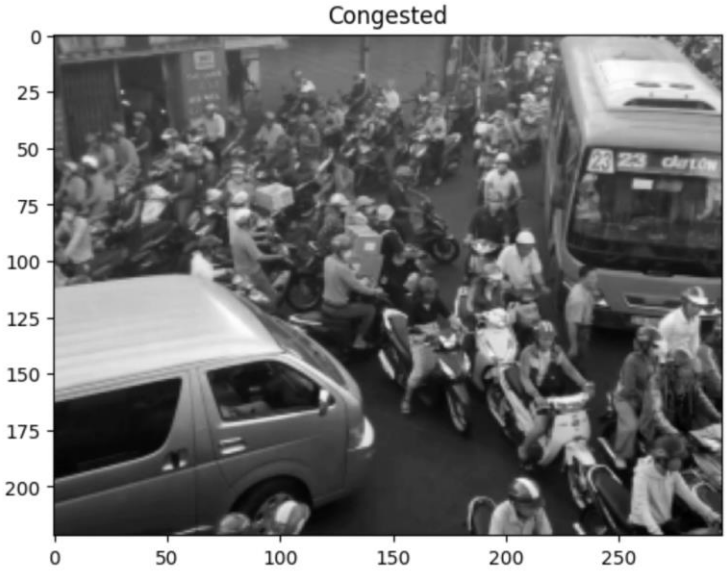


[6]:
```
image_visualizer(X_val, y_val, 17)
```




```
[8]: image_visualizer(X_val, y_val, 171)
```

```
]: image_visualizer(X_val, y_val, 171)
```



]:
```
image_visualizer(X_val, y_val, 117)
```

Conclusion

In conclusion, the utilization of deep learning technology for real-time traffic congestion avoidance at high-density urban intersections represents a significant advancement in intelligent transportation systems. This innovative approach harnesses the power of deep learning algorithms to process captured images swiftly and effectively, providing timely insights into traffic conditions.

References

1. Manu S. Pillai, Gopal Chaudhary, Manju Khari, Rube´n Gonza´lez Crespo (2021). Real-time image enhancement for an automatic automobile accident detection through CCTV using deep learning. Soft Computing (2021) 25:11929–11940
2. Da Yang, Yuezhu Wu, Feng Sun, Jing Chen, Donghai Zhai, Chuanyun Fu (2021). Freeway accident detection and classification based on the multi-vehicle trajectory data and deep learning model. Transportation Research Part C, 103303-103325
3. Sumit Mishra, Praveen Kumar Rajendran, Luiz Felipe Vecchietti, Dongsoo Har (2021). Sensing Accident-Prone Features in Urban Scenes for Proactive Driving and Accident Prevention, 9401-9414.
4. Xueyan Yin, Genze Wu, Jinze Wei, Yanming Shen, Heng Qi (2021). Deep Learning on Traffic Prediction: Methods, Analysis, and Future Directions. IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 23, NO. 6, 4927-4943
5. Ying Gao, Jinlong Li, Zhigang Xu, Zhangqi Liu, Xiangmo Zhao, Jianhua Chen (2021). A novel image-based convolutional neural network approach for traffic congestion estimation. Expert Systems with Applications 180 (2021), 115037-115050
6. G. S. R. Satyanarayana, Sudhan Majhi, Santos Kumar Das (2021). A Vehicle Detection Technique Using Binary Images for Heterogeneous and Lane-Less Traffic. IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 70, 5007514
7. Yue Tu, Shukuan Lin, Jianzhong Qiao, Bin Liu (2021). Deep traffic congestion prediction model based on road segment grouping. Applied Intelligence (2021) 51:8519–8541
8. Ousmane Youme, Theophile Bayet, Jean Marie Dembele, Christophe Cambier (2021). Deep Learning and Remote Sensing: Detection of Dumping Waste Using UAV. Procedia Computer Science 185 (2021), 361–369
9. Noureen Zafar, Irfan Ul Haq, Huniya Sohail, Jawad-Ur-Rehman, Chughtai, Muhammad Muneeb (2022). Traffic Prediction in Smart Cities Based on Hybrid Feature Space. Traffic Prediction in Smart Cities Based on Hybrid Feature Space, 134333-134328
10. Himanshu Gupta, Om Prakash Verma (2021). Monitoring and surveillance of urban road traffic using low altitude drone images: a deep learning approach. Multimedia Tools and Applications (2022) 81:19683–19703
11. Srishti Srivastava, Sarthak Narayan, Sparsh Mittal (2021). A survey of deep learning techniques for vehicle detection from UAV images, Journal of Systems Architecture 117 (2021), 102152
12. Wei Lo, Hamed Alqahtanib, Kutub Thakurc, Ahmad Almadhord, Subhash Chandere, Gulshan Kumar (2022). A hybrid deep learning based intrusion detection system using spatial-temporal representation of in-vehicle network traffic. Vehicular Communications 35 (2022), 100471
13. Sardar Waqar Khan, Qasim Hafeez, Muhammad Irfan Khalid, Roobaea Alroobaea, Saddam Hussain, Jawaid Iqbal, Jasem Almotiri, Syed Sajid Ullah (2022). Anomaly

Detection in Traffic Surveillance Videos Using Deep Learning. *Sensors* 2022, *22*, 6563

14. Kamalakar Vijay Thakare, Debi Prosad Dogra,Heeseung Choi, Haksub Kim, Ig-Jae Kim (2022). Object Interaction-Based Localization and Description of Road Accident Events Using Deep Learning. IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 23, NO. 11, 20601-20613.
15. Hailong Huang, Andrey V. Savkin, Chao Huang (2021). Decentralized Autonomous Navigation of a UAV Network for Road Traffic Monitoring. IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS VOL. 57, NO. 4, 2258-2564.