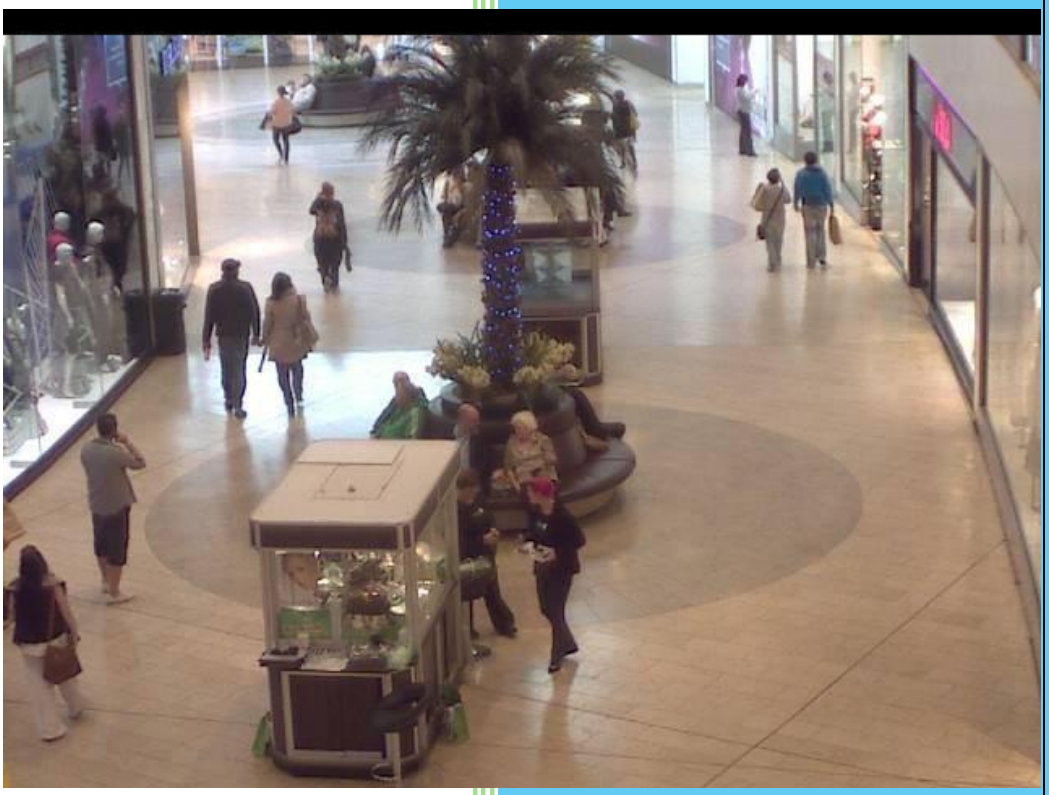
2024

IMPROVE MALL TRAFFIC WITH AI



Contents

NNN – PROJECT SUMMARY

1. Project Overview

Description**:**

The **NNN (**Neural Network Narratives**)** project integrates object detection, data modelling, and time series analysis to analyse people counts in specific zones. It generates actionable insights using deep learning models and offers an interactive UI for visualization and interaction.

2. Team Contributions

Description**:**

Each team member contributed to different aspects of the project:

- **Chandana (Team Leader)** Managed the project, conducted time series analysis, and designed the user interface.
- **Vinay (Data Modeller)** Focused on Objection detection, data cleaning, data Modelling, and insightful visualizations.
- **Devika(Time Series Analyst)** Specialized in advanced time series analysis and forecasting techniques.

3. Project Modules

3.1. NNN_FINAL_OB_TS: Object Detection and Time Series Analysis

Description**:**

This module handles image preprocessing, object detection, and time series analysis:

- **Image Preprocessing:** Images were processed with convolution, padding, and max pooling to enhance feature extraction.
- **Object Detection:** Leveraged **Faster-RCNN** to identify people in images, with utility functions for displaying images, drawing bounding boxes, and counting people.
- **EDA and Visualizations:** Insights like zone-wise people counts, top 20 images by people count, scatter plots, bar charts, and pie charts were generated.

- **Time Series Analysis:** Seasonal decomposition and ARIMA/SARIMA models were applied to forecast trends. Visualizations include ACF/PACF plots and decomposition charts.

3.2. NNN_FINAL_MODEL: Deep Learning Model

Description:

This module predicts people counts and zones from images using deep learning:

- **Custom Image Generator:** Loaded and pre-processed images while generating labels for zone-wise counts.
- **Model Architecture:** Utilized **VGG16** with ReLU activation. Training parameters included a batch size of 16 and 10 epochs.
- **Results:** Achieved low loss and mean absolute error (MAE). The trained models were saved in Keras and H5 formats for reuse.
- **Prediction:** New images were processed for accurate people and zone predictions.

3.3. NNN_FINAL_UI/UX: User Interface

Description:

A user-friendly interface built with **Streamlit** to visualize detection and analysis results:

- **Key Features:**
  - Upload and display images with bounding boxes and people counts.
  - Visualize zone-wise distributions and time series insights.
  - Integrated with **Ngrok** for secure public access.
- **Technology Stack:** Streamlit, PIL, OpenCV, and Ngrok.

---

4. Key Project Outputs

Description:

The project outputs provide actionable insights:

- **Object Detection:** Accurate total and zone-wise counts of people from images.
- **EDA Visualizations:** Scatter plots, pie charts, and bar graphs for zone-wise data.
- **Time Series Insights:** Forecasts of trends and seasonal variations in people counts.

- **Model Predictions:** Deep learning predictions for new datasets.

---

5. Maintenance and Handover Instructions

5.1. Code Repository

Description:
The complete codebase is structured as follows:

- src/: Source code for all modules.
- models/: Saved Keras and H5 models.
- data/: Raw and processed datasets.
- ui/: Streamlit UI code.

5.2. Execution Steps

Description:

- **Object Detection:** Run ob_detection.py for detection and eda_visualizations.py for analysis.
- **Model Training:** Use train_model.py to train the deep learning model.
- **Time Series Analysis:** Execute time_series_analysis.py for decomposition and forecasting.
- **UI:** Launch the Streamlit app using streamlit run app.py and share via Ngrok (ngrok http 8501).

5.3. Dependencies

Description:
Install required libraries with pip install -r requirements.txt. Dependencies include TensorFlow, Streamlit, PIL, OpenCV, Statsmodels, and Matplotlib.

5.4. Model Retraining

Description:
To retrain the model with new data:

- Update the dataset.
- Adjust hyperparameters like batch size and epochs if necessary.
- Save the new model after training.

---

6. Future Recommendations

Description:
Suggestions for improving and extending the project:

- Incorporate real-time video processing for live detection and monitoring.
- Implement anomaly detection to identify unusual activity in zones.
- Use advanced hybrid models like LSTM-ARIMA for better forecasting accuracy.

---

7. Libraries Used

Below is the list of libraries used in the project across its various modules:

Core Python Libraries

- **os**: For handling file paths and directory management.
- **glob**: For retrieving files matching specific patterns.

Data Handling and Processing

- **Pandas**: For data manipulation and analysis (e.g., handling datasets, creating dataframes).
- **Numpy**: For numerical computations and array operations.

Machine Learning and Deep Learning

- **TensorFlow**: Framework for building and training the deep learning models (e.g., VGG16).
- **Keras**: High-level API in TensorFlow for building the deep learning architecture.
- **Scikit-learn**: For statistical analysis, data preprocessing, and model evaluation.

Image Processing and Object Detection

- **OpenCV (cv2)**: For image manipulation (e.g., resizing, bounding box overlays).
- **Pillow (PIL)**: For image loading, editing, and saving.
- **Matplotlib**: For visualizing image-related operations like bounding boxes.

Visualization

- **Matplotlib**: To create bar plots, scatter plots, and decomposition graphs.
- **Seaborn**: For enhanced visualizations like heatmaps, bar plots, and scatter plots.

Time Series Analysis

- **Statsmodels**: For time series decomposition, ARIMA, SARIMA, and statistical tests like Dickey-Fuller.

UI Development

- **Streamlit**: To build the web application interface for visualizing results interactively.

Deployment and Public Access

- **Ngrok**: For securely hosting the Streamlit application for external access.

Miscellaneous

- **Datetime**: For timestamp generation and time series manipulation.
- **Warnings**: To suppress unnecessary warnings during code execution.
