Appendix A: Code Samples

Code Sample 1: Vehicle Cropping Strategy

```
def generate_cnn_training_dataset():
    for each sequence:
        for each frame:
            for each annotated vehicle:
                # 1. Extract bounding box
                crop = frame[y1:y2, x1:x2]

                # 2. Handle edge cases
                crop = clamp_to_image_bounds(crop)

                # 3. Resize to fixed dimensions
                crop = cv2.resize(crop, (64, 64))

                # 4. Normalize to [0,1]
                crop = crop.astype('float32') / 255.0

                # 5. Store with label
                dataset.append((crop, class_idx))
```

Code Sample 2: CNN Architecture

```
VehicleClassifier(
  features: Sequential(
    # Block 1: Initial feature extraction
    Conv2d(3 → 32, kernel=3×3, padding=1)
    BatchNorm2d(32)
    ReLU()
    MaxPool2d(2×2)  # 64×64 → 32×32

    # Block 2: Mid-level features
    Conv2d(32 → 64, kernel=3×3, padding=1)
    BatchNorm2d(64)
    ReLU()
    MaxPool2d(2×2)  # 32×32 → 16×16

    # Block 3: High-level features
    Conv2d(64 → 128, kernel=3×3, padding=1)
    BatchNorm2d(128)
    ReLU()
    MaxPool2d(2×2)  # 16×16 → 8×8

    # Block 4: Abstract features
    Conv2d(128 → 256, kernel=3×3, padding=1)
    BatchNorm2d(256)
    ReLU()
```

```
    MaxPool2d(2×2)  # 8×8 → 4×4
  )

  classifier: Sequential(
    Flatten()  # 256×4×4 = 4,096 features
    Linear(4096 → 256)
    ReLU()
    Dropout(0.5)
    Linear(256 → num_classes)
  )
)
```

Code Sample 3: LMV/HMV Mapping

```
LMV_CLASSES = {'car', 'van', 'others', 'motor'}  # Light Motor Vehicles
HMV_CLASSES = {'bus', 'truck'}                   # Heavy Motor Vehicles

def map_to_lmv_hmv(fine_class: str) -> str:
    if fine_class.lower() in LMV_CLASSES:
        return "LMV"
    elif fine_class.lower() in HMV_CLASSES:
        return "HMV"
    else:
        return "Unknown"
```

Code Sample 4: Traffic Forecasting

```
# Feature engineering: Use previous K frames to predict next frame
K = 5  # History length
for lag in range(1, K+1):
    df[f'total_lag_{lag}'] = df['total_count'].shift(lag)

X = df[[f'total_lag_{i}' for i in range(1, K+1)]].values
y = df['total_count'].values

# Train/test split (temporal, no shuffling)
split_idx = int(0.8 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

# Random Forest Regressor
model = RandomForestRegressor(n_estimators=200, random_state=42)
model.fit(X_train, y_train)
```

01_dataset_exploration

December 6, 2025

1 AAI-521 Final Project – Group 3

1.1 01 – DETRAC Dataset Exploration

Goal:
Understand the UA-DETRAC dataset structure and annotations by: - Verifying folder paths and files - Parsing XML annotation files - Visualizing sample frames with bounding boxes - Exploring basic statistics of bounding boxes and frames

This notebook is exploratory only. No model training happens here.

Imports & Configuration

[1]:
```
import os
import sys
from pathlib import Path
import cv2
import numpy as np
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET

# For pretty plots
plt.style.use("seaborn-v0_8")

# Root of the DETRAC dataset (adapt to your structure)
PROJECT_ROOT = Path().resolve().parent  # project root if notebooks/ is one␣
 ↪level down
DATA_ROOT = PROJECT_ROOT / "data"

IMAGES_ROOT = DATA_ROOT / "DETRAC-Images"
TRAIN_ANN_ROOT = DATA_ROOT / "DETRAC-Train-Annotations"
TEST_ANN_ROOT = DATA_ROOT / "DETRAC-Test-Annotations"
SRC_DIR = PROJECT_ROOT / "src"

if str(SRC_DIR) not in sys.path:
    sys.path.append(str(SRC_DIR))

from utils_detrac import load_detrac_annotations, VehicleClassifier,␣
 ↪predict_vehicle_class
```

```
print("Images root:", IMAGES_ROOT)
print("Train annotations root:", TRAIN_ANN_ROOT)
print("Test annotations root:", TEST_ANN_ROOT)

# Choose one sequence for detailed exploration
SEQ_ID = "MVI_20011"
SEQ_IMAGES_DIR = IMAGES_ROOT / SEQ_ID
SEQ_TRAIN_ANN_FILE = TRAIN_ANN_ROOT / f"{SEQ_ID}.xml"
```

```
Images root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Images
Train annotations root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Train-Annotations
Test annotations root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Test-Annotations
```

Sanity Checks

[2]:
```
assert IMAGES_ROOT.exists(), f"Images folder not found: {IMAGES_ROOT}"
assert TRAIN_ANN_ROOT.exists(), f"Train annotations folder not found:␣
 ↪{TRAIN_ANN_ROOT}"

assert SEQ_IMAGES_DIR.exists(), f"Sequence folder not found: {SEQ_IMAGES_DIR}"
assert SEQ_TRAIN_ANN_FILE.exists(), f"Annotation XML not found:␣
 ↪{SEQ_TRAIN_ANN_FILE}"

print("All key DETRAC paths exist  ")
```

```
All key DETRAC paths exist
```

1.2 1. Parse DETRAC XML Annotations

We load the UA-DETRAC XML format and convert it into a Python-friendly structure:

- Key: frame index (int)
- Value: list of dicts with:
  - `id` – vehicle track ID
  - `bbox` – `[left, top, width, height]`
  - `class` – vehicle type (car, bus, van, truck, etc.)

Load One Sequence & Basic Info

[3]:
```
annotations = load_detrac_annotations(SEQ_TRAIN_ANN_FILE)

image_files = sorted(
    f for f in os.listdir(SEQ_IMAGES_DIR) if f.lower().endswith(".jpg")
)
frame_numbers = [int(f.replace("img", "").replace(".jpg", "")) for f in␣
 ↪image_files]
```

```
print(f"Found {len(image_files)} image frames in {SEQ_IMAGES_DIR.name}")
print("First 5 frames:", image_files[:5])
```

```
Found 664 image frames in MVI_20011
First 5 frames: ['img00001.jpg', 'img00002.jpg', 'img00003.jpg', 'img00004.jpg',
'img00005.jpg']
```

1.3 2. Visualize a Sample Frame with Bounding Boxes

We overlay the bounding boxes onto a sample frame to confirm the annotation format and alignment with the images.

Dataset Statistics

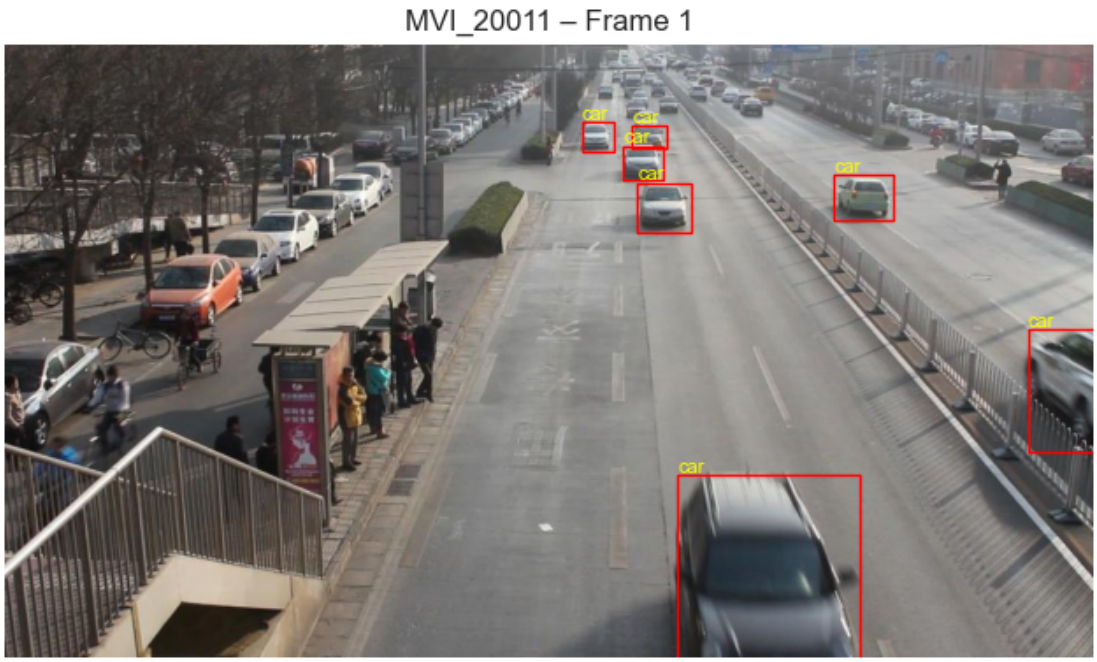
[4]:
```
image_files = sorted([f for f in os.listdir(SEQ_IMAGES_DIR) if f.endswith(".
 ↪jpg")])

print("Dataset Statistics")
print("------------------")
print(f"- Sequence ID: {SEQ_ID}")
print(f"- Images folder: {SEQ_IMAGES_DIR}")
print(f"- Total frames (image files): {len(image_files)}")
print(f"- Annotated frames: {len(annotations)}")

# Sample image properties (height, width, channels)
if image_files:
    sample_path = SEQ_IMAGES_DIR / image_files[0]
    sample_img = cv2.imread(str(sample_path))
    if sample_img is not None:
        h, w, c = sample_img.shape
        print(f"- Sample image shape: {h} x {w} x {c}")
    else:
        print("- WARNING: Could not read sample image to get shape.")
else:
    print("- WARNING: No .jpg files found in images directory.")

# Simple vehicle-per-frame stats using existing annotations
vehicle_counts = [len(targets) for _, targets in annotations.items()]
if vehicle_counts:
    print(f"- Average vehicles per annotated frame: {np.mean(vehicle_counts):.
 ↪2f}")
    print(f"- Max vehicles in a frame: {np.max(vehicle_counts)}")
    print(f"- Min vehicles in a frame: {np.min(vehicle_counts)}")
else:
    print("- No vehicles found in annotations.")
```

```
Dataset Statistics
------------------
```

```
- Sequence ID: MVI_20011
- Images folder: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Images/MVI_20011
- Total frames (image files): 664
- Annotated frames: 664
- Sample image shape: 540 x 960 x 3
- Average vehicles per annotated frame: 11.53
- Max vehicles in a frame: 16
- Min vehicles in a frame: 6
```

Visualization Helper

[5]:
```python
def show_frame_with_boxes(seq_images_dir, annotations, frame_num):
    img_file = f"img{frame_num:05d}.jpg"
    img_path = seq_images_dir / img_file

    if not img_path.exists():
        print(f"Image not found for frame {frame_num}: {img_path}")
        return

    img = cv2.imread(str(img_path))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    fig, ax = plt.subplots(figsize=(8, 6))
    ax.imshow(img)
    ax.set_title(f"{seq_images_dir.name} - Frame {frame_num}")

    for target in annotations.get(frame_num, []):
        x, y, w, h = target["bbox"]
        rect = plt.Rectangle((x, y), w, h, fill=False, color="red", linewidth=1)
        ax.add_patch(rect)
        ax.text(x, y - 2, target["class"], color="yellow", fontsize=8)

    ax.axis("off")
    plt.show()

show_frame_with_boxes(SEQ_IMAGES_DIR, annotations, frame_num=1)
```



1.4 3. Visualize Multiple Frames

We next show several frames in a grid to get a qualitative feel for traffic density, camera angle, and vehicle variety.

Multiple Frames Grid

[6]:
```
def show_frames_grid(seq_images_dir, annotations, frame_indices, rows=2,␣
↪cols=3):
    fig, axes = plt.subplots(rows, cols, figsize=(15, 8))
    axes = axes.flatten()

    for ax, fidx in zip(axes, frame_indices):
        img_file = f"img{fidx:05d}.jpg"
        img_path = seq_images_dir / img_file

        if not img_path.exists():
            ax.axis("off")
            continue

        img = cv2.imread(str(img_path))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        ax.imshow(img)
        ax.set_title(f"Frame {fidx}")
```

```
        for target in annotations.get(fidx, []):
            x, y, w, h = target["bbox"]
            rect = plt.Rectangle((x, y), w, h, fill=False, color="lime",␣
 ↪linewidth=0.8)
            ax.add_patch(rect)

        ax.axis("off")

    plt.tight_layout()
    plt.show()

sample_indices = [1, 10, 25, 50, 75, 100]
show_frames_grid(SEQ_IMAGES_DIR, annotations, sample_indices)
```

1.5 4. Bounding Box Statistics

We summarize the distribution of vehicle bounding boxes in one sequence:

- Number of vehicles per frame
- Bounding box width/height
- Bounding box area
- Vehicle class frequencies

These statistics will appear in the report's EDA section.

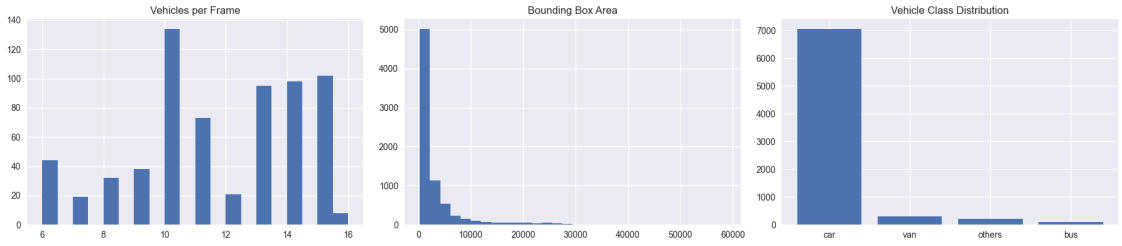
Bounding Box Statistics

[7]:
```
all_widths, all_heights, all_areas = [], [], []
vehicles_per_frame = []
class_counts = {}

for frame_num, targets in annotations.items():
```

```
    vehicles_per_frame.append(len(targets))
    for t in targets:
        x, y, w, h = t["bbox"]
        all_widths.append(w)
        all_heights.append(h)
        all_areas.append(w * h)
        cls = t["class"]
        class_counts[cls] = class_counts.get(cls, 0) + 1

# ----------------------------
# Aspect ratios (W/H)
# ----------------------------
aspect_ratios = []
for w, h in zip(all_widths, all_heights):
    if h > 0:
        aspect_ratios.append(w / h)

# ----------------------------
# Robust print section here
# ----------------------------
print("Frames with vehicles:", len(vehicles_per_frame))
print("Mean vehicles per frame:", np.mean(vehicles_per_frame))

if len(all_widths) > 0:
    print("Bounding box width (mean, std):", np.mean(all_widths), np.
 ↪std(all_widths))
    print("Bounding box height (mean, std):", np.mean(all_heights), np.
 ↪std(all_heights))
    print("Bounding box area (mean, std):", np.mean(all_areas), np.
 ↪std(all_areas))
    if aspect_ratios:
        print(
            "Bounding box aspect ratio W/H (mean, std):",
            np.mean(aspect_ratios),
            np.std(aspect_ratios),
        )
else:
    print("No bounding boxes found - check annotation parsing.")

print("Vehicle class counts:", class_counts)

# ----------------------------
# Plots (only shown if data exists)
# ----------------------------
if len(all_widths) > 0:
    fig, axes = plt.subplots(1, 3, figsize=(18, 4))
    axes[0].hist(vehicles_per_frame, bins=20)
```

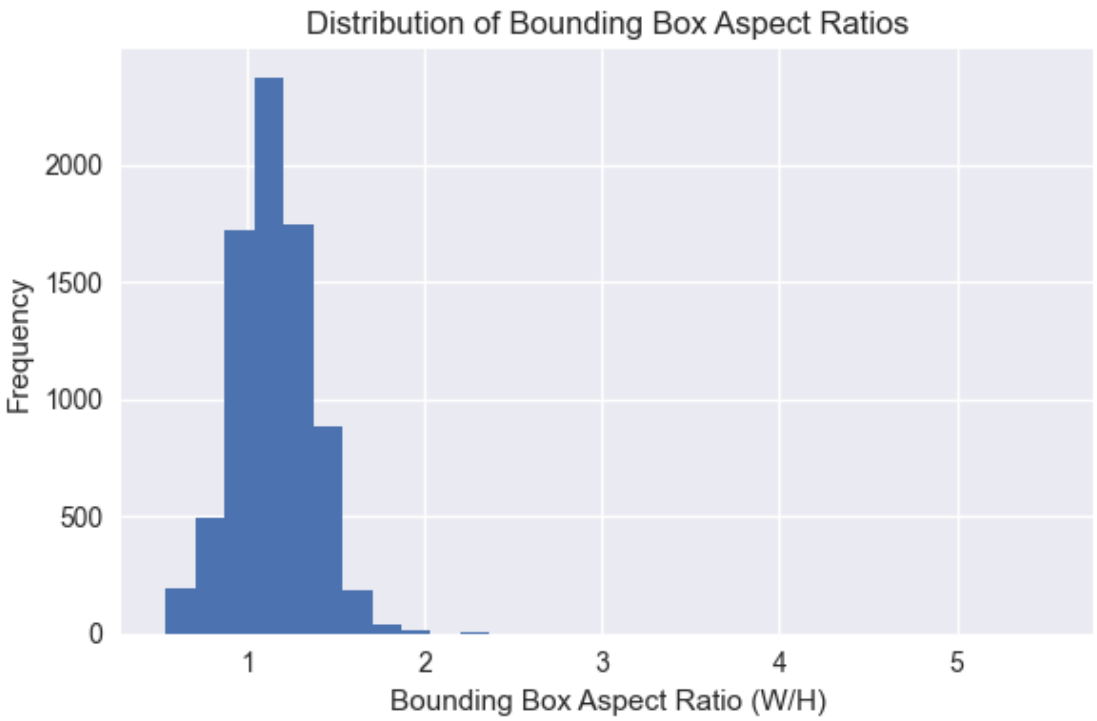
```
    axes[0].set_title("Vehicles per Frame")

    axes[1].hist(all_areas, bins=30)
    axes[1].set_title("Bounding Box Area")

    axes[2].bar(class_counts.keys(), class_counts.values())
    axes[2].set_title("Vehicle Class Distribution")

    plt.tight_layout()
    plt.show()
else:
    print("Skipping plots because no bounding boxes were parsed.")
```

```
Frames with vehicles: 664
Mean vehicles per frame: 11.528614457831326
Bounding box width (mean, std): 50.59611234487263 32.04088096966129
Bounding box height (mean, std): 46.65976864794252 35.22487032706986
Bounding box area (mean, std): 3392.733557634226 5752.800977299735
Bounding box aspect ratio W/H (mean, std): 1.1492786235200811
0.22514196786078045
Vehicle class counts: {'car': 7053, 'van': 296, 'others': 211, 'bus': 95}
```



Bounding box aspect ratio distribution (W/H)

[8]:
```
if "aspect_ratios" in globals() and len(aspect_ratios) > 0:
    plt.figure(figsize=(6, 4))
    plt.hist(aspect_ratios, bins=30)
    plt.xlabel("Bounding Box Aspect Ratio (W/H)")
    plt.ylabel("Frequency")
    plt.title("Distribution of Bounding Box Aspect Ratios")
    plt.tight_layout()
    plt.show()
else:
    print("No aspect ratios to plot (no bounding boxes or all heights = 0).")
```



1.6 5. Summary

- The DETRAC dataset is correctly organized under `data/DETRAC-*`.
- We confirmed that:
  - XML annotations align with the images.
  - Bounding boxes are reasonable in size.
  - Vehicle classes include: car, bus, van, truck, and others.
- These insights will guide our preprocessing and model design in the next notebooks.

Next step: create a cropped vehicle dataset suitable for training a CNN classifier.

02_preprocessing_and_cropping

December 6, 2025

1 AAI-521 Final Project – Group 3

1.1 02 – Preprocessing and Vehicle Cropping

Goal:
Convert UA-DETRAC sequences into a compact vehicle classification dataset by:

- Iterating over training sequences
- Cropping each annotated vehicle
- Resizing crops to a fixed resolution
- Encoding classes as integer labels
- Saving the resulting dataset to disk for later model training

This notebook **produces the training dataset** used by the CNN model.

Imports & Configuration

[1]:
```
import os
import sys
from pathlib import Path
import cv2
import numpy as np
import random
import xml.etree.ElementTree as ET
from tqdm import tqdm
import matplotlib.pyplot as plt

PROJECT_ROOT = Path().resolve().parent
DATA_ROOT = PROJECT_ROOT / "data"

IMAGES_ROOT = DATA_ROOT / "DETRAC-Images"
TRAIN_ANN_ROOT = DATA_ROOT / "DETRAC-Train-Annotations"

TARGET_SIZE = (64, 64)  # (width, height) for CNN inputs
DATA_PATH = DATA_ROOT / "cropped_vehicle_dataset.npz"
SRC_DIR = PROJECT_ROOT / "src"

if str(SRC_DIR) not in sys.path:
    sys.path.append(str(SRC_DIR))
```

```
from utils_detrac import load_detrac_annotations, VehicleClassifier,␣
 ↪predict_vehicle_class

print("Images root:", IMAGES_ROOT)
print("Train annotations root:", TRAIN_ANN_ROOT)
print("Output dataset path:", DATA_PATH)
```

```
Images root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Images
Train annotations root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Train-Annotations
Output dataset path: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/cropped_vehicle_dataset.npz
```

1.2 1. Generate Cropped Vehicle Dataset

We iterate over all training XML files, match them to image folders, and generate:

- `images`: array of shape `(N, H, W, 3)` in `[0, 1]` range
- `labels`: integer class IDs
- `class_to_idx`: mapping from vehicle type string → integer

The resulting arrays are saved to `cropped_vehicle_dataset.npz`.

Dataset Generation Function

[2]:
```
def generate_cnn_training_dataset(
    annotations_root: Path,
    images_root: Path,
    target_size=TARGET_SIZE,
):

    all_images = []
    all_labels = []
    all_metadata = []

    class_to_idx = {}
    next_class_idx = 0

    xml_files = sorted(
        f for f in annotations_root.iterdir()
        if f.suffix.lower() == ".xml"
    )

    for xml_path in tqdm(xml_files, desc="Sequences"):
        seq_id = xml_path.stem  # e.g., "MVI_20011"
        seq_images_dir = images_root / seq_id
        if not seq_images_dir.exists():
            print(f"[WARN] No image folder for {seq_id}, skipping.")
            continue
```

```
    annotations = load_detrac_annotations(xml_path)

    for frame_num, targets in annotations.items():
        img_file = seq_images_dir / f"img{frame_num:05d}.jpg"
        if not img_file.exists():
            continue

        frame = cv2.imread(str(img_file))
        if frame is None:
            continue

        h_img, w_img = frame.shape[:2]

        for t in targets:
            x, y, w, h = t["bbox"]
            cls = t["class"]

            # Clamp bounding box to image bounds
            x1 = max(int(x), 0)
            y1 = max(int(y), 0)
            x2 = min(int(x + w), w_img)
            y2 = min(int(y + h), h_img)

            if x2 <= x1 or y2 <= y1:
                continue

            crop = frame[y1:y2, x1:x2]
            crop = cv2.resize(crop, target_size)
            crop = cv2.cvtColor(crop, cv2.COLOR_BGR2RGB)
            crop = crop.astype("float32") / 255.0

            if cls not in class_to_idx:
                class_to_idx[cls] = next_class_idx
                next_class_idx += 1

            label_idx = class_to_idx[cls]

            all_images.append(crop)
            all_labels.append(label_idx)
            all_metadata.append({
                "seq_id": seq_id,
                "frame_num": frame_num,
                "class": cls,
                # NEW: useful extra info, inspired by base_model
                "image_path": str(img_file),
                "bbox": (int(x1), int(y1), int(x2), int(y2)),
```

```
                    "label_idx": int(label_idx),
                })


    images = np.stack(all_images, axis=0)
    labels = np.array(all_labels, dtype=np.int64)

    print("Dataset size:", images.shape[0])
    print("Classes:", class_to_idx)

    return images, labels, class_to_idx, all_metadata
```

Run Generation & Save

[3]:
```
# Make sure DATA_PATH is a Path
DATA_PATH = Path(DATA_PATH)

if DATA_PATH.exists():
    print(f" Cropped dataset already exists at {DATA_PATH}. Skipping␣
 ↪generation.")
else:
    images, labels, class_to_idx, metadata = generate_cnn_training_dataset(
        TRAIN_ANN_ROOT,
        IMAGES_ROOT,
        target_size=TARGET_SIZE,
    )

    np.savez_compressed(
        DATA_PATH,
        images=images,
        labels=labels,
        class_to_idx=class_to_idx,
        metadata=np.array(metadata, dtype=object),
    )

    print(f" Saved cropped dataset to {DATA_PATH}")
```

```
 Cropped dataset already exists at
/Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/cropped_vehicle_dataset.npz.
Skipping generation.
```

1.3 2. Quick Sanity Check on the Cropped Dataset

We visualize a few random crops with their labels to confirm the dataset looks correct.

Quick Sanity Check

[5]:
```python
# -------------------------
# Load the dataset
# -------------------------
if DATA_PATH.exists():
    print(f"Loading dataset from {DATA_PATH}")
    data = np.load(DATA_PATH, allow_pickle=True)
    images = data["images"]
    labels = data["labels"]
    class_to_idx = data["class_to_idx"].item()
    metadata = data["metadata"]
else:
    raise FileNotFoundError(
        f"{DATA_PATH} does not exist. Run the generation cell first."
    )

# Reverse lookup: index → class_name
idx_to_class = {v: k for k, v in class_to_idx.items()}

# -------------------------
# Show sample images
# -------------------------
n_show = 8
indices = random.sample(range(len(images)), k=n_show)

fig, axes = plt.subplots(2, 4, figsize=(12, 6))
axes = axes.flatten()

for ax, i in zip(axes, indices):
    ax.imshow(images[i])
    label = labels[i]
    ax.set_title(idx_to_class[label])
    ax.axis("off")

plt.tight_layout()
plt.show()
```

```
Loading dataset from /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/cropped_vehicle_dataset.npz
```

1.4 3. Summary

- Generated a cropped vehicle classification dataset from UA-DETRAC:
  - Total samples: `N =` *(see above output)*
  - Image size: `64 × 64 × 3`
  - Classes: car, bus, van, truck, etc.
- Saved the dataset to `data/cropped_vehicle_dataset.npz`.

Next step: train a CNN classifier using this dataset (Notebook 03).

03_vehicle_classification_model

December 6, 2025

1 AAI-521 Final Project – Group 3

1.1 03 – Vehicle Classification Model (LMV vs HMV)

Goal:
Train and evaluate a convolutional neural network (CNN) using the cropped vehicle dataset created in Notebook 02.

This notebook includes: - Loading preprocessed crops and labels - Train/validation split - CNN architecture - Training loop and learning curves - Evaluation (accuracy, classification report, confusion matrix) - Qualitative results (annotated frames and video)

Imports & Configuration

[ ]:
```
import sys
import numpy as np
from pathlib import Path

import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader, random_split

import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

PROJECT_ROOT = Path().resolve()
DATA_ROOT = PROJECT_ROOT / "data"
IMAGES_ROOT = DATA_ROOT / "DETRAC-Images"
TRAIN_ANN_ROOT = DATA_ROOT / "DETRAC-Train-Annotations"
CROPPED_DATA_PATH = DATA_ROOT / "cropped_vehicle_dataset.npz"
SRC_DIR = PROJECT_ROOT / "src"

if str(SRC_DIR) not in sys.path:
    sys.path.append(str(SRC_DIR))

from utils_detrac import load_detrac_annotations, VehicleClassifier,␣
 ↪predict_vehicle_class

if torch.cuda.is_available():
```

```
    DEVICE = torch.device("cuda")
elif torch.backends.mps.is_available():
    DEVICE = torch.device("mps")
else:
    DEVICE = torch.device("cpu")

print("Images root:", IMAGES_ROOT)
print("Train annotations root:", TRAIN_ANN_ROOT)
print("Using device:", DEVICE)
print("Loading dataset from:", CROPPED_DATA_PATH)
```

```
Images root: /Users/victorhugogermano/Development/aai-521-final-
project-g3/data/DETRAC-Images
Train annotations root: /Users/victorhugogermano/Development/aai-521-final-
project-g3/data/DETRAC-Train-Annotations
Using device: mps
Loading dataset from: /Users/victorhugogermano/Development/aai-521-final-
project-g3/outputs/cropped_vehicle_dataset.npz
```

Load Dataset

[5]:
```
data = np.load(CROPPED_DATA_PATH, allow_pickle=True)
images = data["images"]  # (N, H, W, 3), float32 in [0,1]
labels = data["labels"]  # (N,)
class_to_idx = data["class_to_idx"].item()
metadata = data["metadata"]

idx_to_class = {v: k for k, v in class_to_idx.items()}

print("Images shape:", images.shape)
print("Labels shape:", labels.shape)
print("Classes:", idx_to_class)
```

```
Images shape: (598281, 64, 64, 3)
Labels shape: (598281,)
Classes: {0: 'car', 1: 'van', 2: 'others', 3: 'bus'}
```

1.2 1. Train / Validation Split

We split the cropped dataset into training and validation sets using an 80/20 random split with a fixed random seed for reproducibility.

Prepare Tensors & Split

[6]:
```
X = torch.from_numpy(images).permute(0, 3, 1, 2)  # (N,3,H,W)
y = torch.from_numpy(labels)

dataset = TensorDataset(X, y)

train_ratio = 0.8
```

```
n_total = len(dataset)
n_train = int(train_ratio * n_total)
n_val = n_total - n_train

torch.manual_seed(42)
train_dataset, val_dataset = random_split(dataset, [n_train, n_val])

BATCH_SIZE = 128

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Train size: {len(train_dataset)}, Val size: {len(val_dataset)}")
```

```
Train size: 478624, Val size: 119657
```

1.3 2. CNN Architecture

We use a simple convolutional neural network with four convolutional blocks followed by two fully connected layers.

This is our **baseline** model for vehicle classification.

CNN Model Definition

[7]:
```
num_classes = len(class_to_idx)
model = VehicleClassifier(num_classes=num_classes).to(DEVICE)

print(model)
```

```
VehicleClassifier(
  (features): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (6): ReLU()
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (10): ReLU()
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (14): ReLU()
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=4096, out_features=256, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=4, bias=True)
  )
)
```

Training Setup

[8]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

NUM_EPOCHS = 15

history = {
    "train_loss": [],
    "val_loss": [],
    "train_acc": [],
    "val_acc": [],
}
best_val_acc = 0.0
best_state_dict = None
```

1.4 3. Training Loop

We train for a fixed number of epochs and track:

- Training and validation loss
- Training and validation accuracy

We keep the model weights that achieved the best validation accuracy.

Training Loop

[9]:
```
for epoch in range(1, NUM_EPOCHS + 1):
    # ---- Training ----
    model.train()
    train_loss = 0.0
    train_correct = 0
    n_train = 0
```

```
    for xb, yb in train_loader:
        xb = xb.to(DEVICE)
        yb = yb.to(DEVICE)

        optimizer.zero_grad()
        logits = model(xb)
        loss = criterion(logits, yb)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * xb.size(0)
        preds = logits.argmax(dim=1)
        train_correct += (preds == yb).sum().item()
        n_train += xb.size(0)

    train_loss /= n_train
    train_acc = train_correct / n_train

    # ---- Validation ----
    model.eval()
    val_loss = 0.0
    val_correct = 0
    n_val = 0

    with torch.no_grad():
        for xb, yb in val_loader:
            xb = xb.to(DEVICE)
            yb = yb.to(DEVICE)

            logits = model(xb)
            loss = criterion(logits, yb)

            val_loss += loss.item() * xb.size(0)
            preds = logits.argmax(dim=1)
            val_correct += (preds == yb).sum().item()
            n_val += xb.size(0)

    val_loss /= n_val
    val_acc = val_correct / n_val

    history["train_loss"].append(train_loss)
    history["val_loss"].append(val_loss)
    history["train_acc"].append(train_acc)
    history["val_acc"].append(val_acc)

    if val_acc > best_val_acc:
        best_val_acc = val_acc
```

```
        best_state_dict = model.state_dict()

    print(
        f"Epoch {epoch:02d}/{NUM_EPOCHS} "
        f"- train_loss: {train_loss:.4f}, train_acc: {train_acc:.3f} "
        f"- val_loss: {val_loss:.4f}, val_acc: {val_acc:.3f}"
    )

print(f"Best validation accuracy: {best_val_acc:.3f}")
model.load_state_dict(best_state_dict)
```

```
Epoch 01/15 - train_loss: 0.1215, train_acc: 0.960 - val_loss: 0.0835, val_acc:
0.971
Epoch 02/15 - train_loss: 0.0473, train_acc: 0.984 - val_loss: 0.0400, val_acc:
0.986
Epoch 03/15 - train_loss: 0.0288, train_acc: 0.990 - val_loss: 0.0280, val_acc:
0.990
Epoch 04/15 - train_loss: 0.0200, train_acc: 0.993 - val_loss: 0.0225, val_acc:
0.993
Epoch 05/15 - train_loss: 0.0152, train_acc: 0.995 - val_loss: 0.0158, val_acc:
0.995
Epoch 06/15 - train_loss: 0.0125, train_acc: 0.996 - val_loss: 0.0132, val_acc:
0.996
Epoch 07/15 - train_loss: 0.0104, train_acc: 0.997 - val_loss: 0.0136, val_acc:
0.996
Epoch 08/15 - train_loss: 0.0089, train_acc: 0.997 - val_loss: 0.0149, val_acc:
0.995
Epoch 09/15 - train_loss: 0.0078, train_acc: 0.997 - val_loss: 0.0119, val_acc:
0.996
Epoch 10/15 - train_loss: 0.0070, train_acc: 0.998 - val_loss: 0.0099, val_acc:
0.997
Epoch 11/15 - train_loss: 0.0064, train_acc: 0.998 - val_loss: 0.0119, val_acc:
0.997
Epoch 12/15 - train_loss: 0.0056, train_acc: 0.998 - val_loss: 0.0134, val_acc:
0.996
Epoch 13/15 - train_loss: 0.0052, train_acc: 0.998 - val_loss: 0.0177, val_acc:
0.995
Epoch 14/15 - train_loss: 0.0047, train_acc: 0.999 - val_loss: 0.0107, val_acc:
0.997
Epoch 15/15 - train_loss: 0.0045, train_acc: 0.999 - val_loss: 0.0145, val_acc:
0.996
Best validation accuracy: 0.997
```

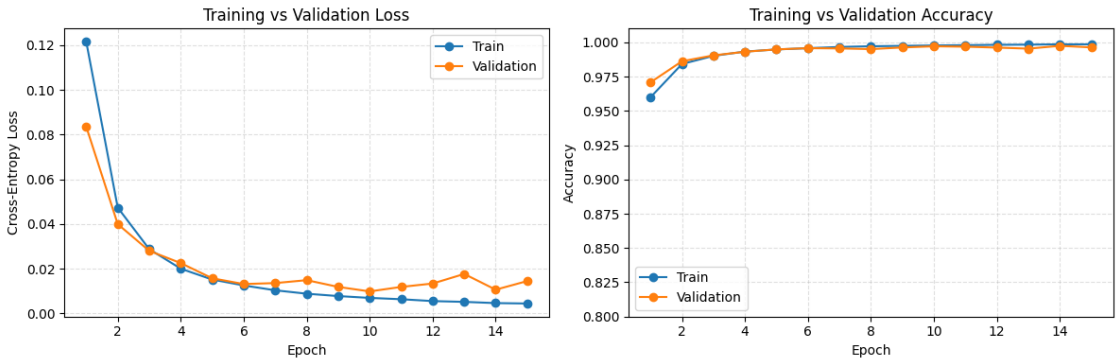
[9]: <All keys matched successfully>

1.5 4. Learning Curves

We plot training and validation loss and accuracy across epochs. These figures will be included in the final report.

Learning Curves

[10]:
```
epochs = range(1, NUM_EPOCHS + 1)

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# ----- LOSS PLOT -----
axes[0].plot(epochs, history["train_loss"], label="Train", marker="o")
axes[0].plot(epochs, history["val_loss"], label="Validation", marker="o")
axes[0].set_title("Training vs Validation Loss")
axes[0].set_xlabel("Epoch")
axes[0].set_ylabel("Cross-Entropy Loss")
axes[0].legend()
axes[0].grid(True, linestyle="--", alpha=0.4)

# ----- ACCURACY PLOT -----
axes[1].plot(epochs, history["train_acc"], label="Train", marker="o")
axes[1].plot(epochs, history["val_acc"], label="Validation", marker="o")
axes[1].set_title("Training vs Validation Accuracy")
axes[1].set_xlabel("Epoch")
axes[1].set_ylabel("Accuracy")
axes[1].legend()
axes[1].set_ylim(0.8, 1.01)  # adjust if needed
axes[1].grid(True, linestyle="--", alpha=0.4)

plt.tight_layout()
plt.show()
```



Save Model

[11]:
```
MODELS_DIR = PROJECT_ROOT / "models"
MODELS_DIR.mkdir(exist_ok=True)

MODEL_PATH = MODELS_DIR / "vehicle_classifier.pth"
torch.save(model.state_dict(), MODEL_PATH)
print("Saved best model to:", MODEL_PATH)
```

```
Saved best model to: /Users/victorhugogermano/Development/aai-521-final-
project-g3/models/vehicle_classifier.pth
```

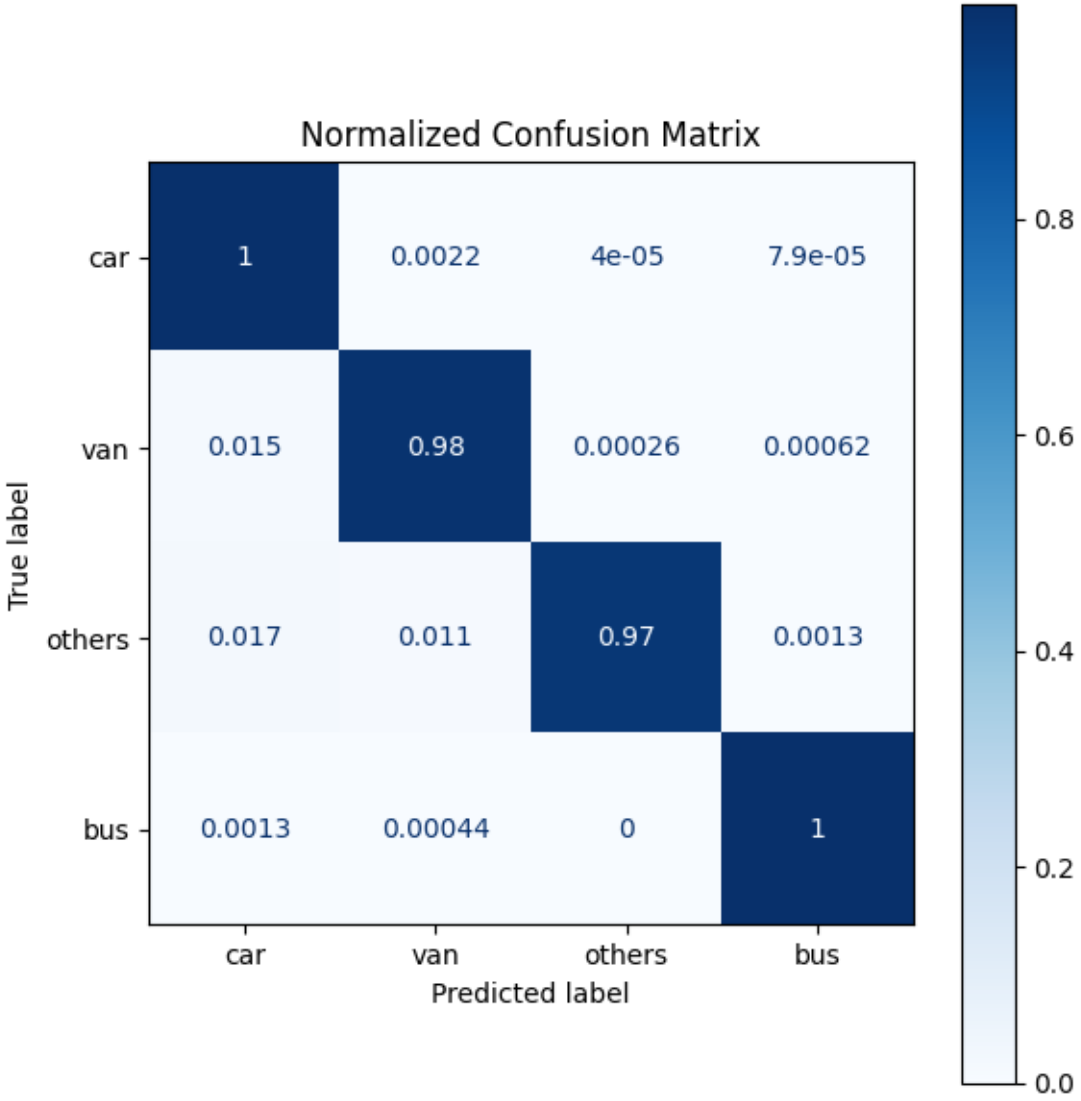
1.6 5. Evaluation on Validation Set

We compute: - Overall accuracy - Classification report (precision, recall, F1) - Normalized confusion matrix

Evaluation

[12]:
```
model.eval()
all_preds = []
all_targets = []

with torch.no_grad():
    for xb, yb in val_loader:
        xb = xb.to(DEVICE)
        logits = model(xb)
        preds = logits.argmax(dim=1).cpu().numpy()
        all_preds.append(preds)
        all_targets.append(yb.numpy())

all_preds = np.concatenate(all_preds)
all_targets = np.concatenate(all_targets)

print(classification_report(
    all_targets, all_preds,
    target_names=[idx_to_class[i] for i in range(num_classes)]
))

fig, ax = plt.subplots(figsize=(6, 6))
ConfusionMatrixDisplay.from_predictions(
    all_targets, all_preds,
    display_labels=[idx_to_class[i] for i in range(num_classes)],
    normalize="true",
    ax=ax,
    cmap="Blues",
)
ax.set_title("Normalized Confusion Matrix")
plt.tight_layout()
plt.show()
```

	precision	recall	f1-score	support
car	1.00	1.00	1.00	100749
van	0.98	0.98	0.98	11327
others	0.99	0.97	0.98	749
bus	1.00	1.00	1.00	6832
accuracy			1.00	119657
macro avg	0.99	0.99	0.99	119657
weighted avg	1.00	1.00	1.00	119657



1.7 6. Qualitative Results – Annotated Frames

We apply the trained CNN on original DETRAC frames to visualize its predictions per vehicle bounding box.

Imports + paths

[13]:
```
import cv2
import xml.etree.ElementTree as ET

# define TARGET_SIZE using the cropped dataset shape (to match training)
# images: (N, H, W, 3)
crop_h, crop_w = images.shape[1], images.shape[2]
TARGET_SIZE = (crop_w, crop_h)  # (width, height) for cv2.resize
print("CNN input size (W,H):", TARGET_SIZE)
```

```
CNN input size (W,H): (64, 64)
```

annotate a single frame

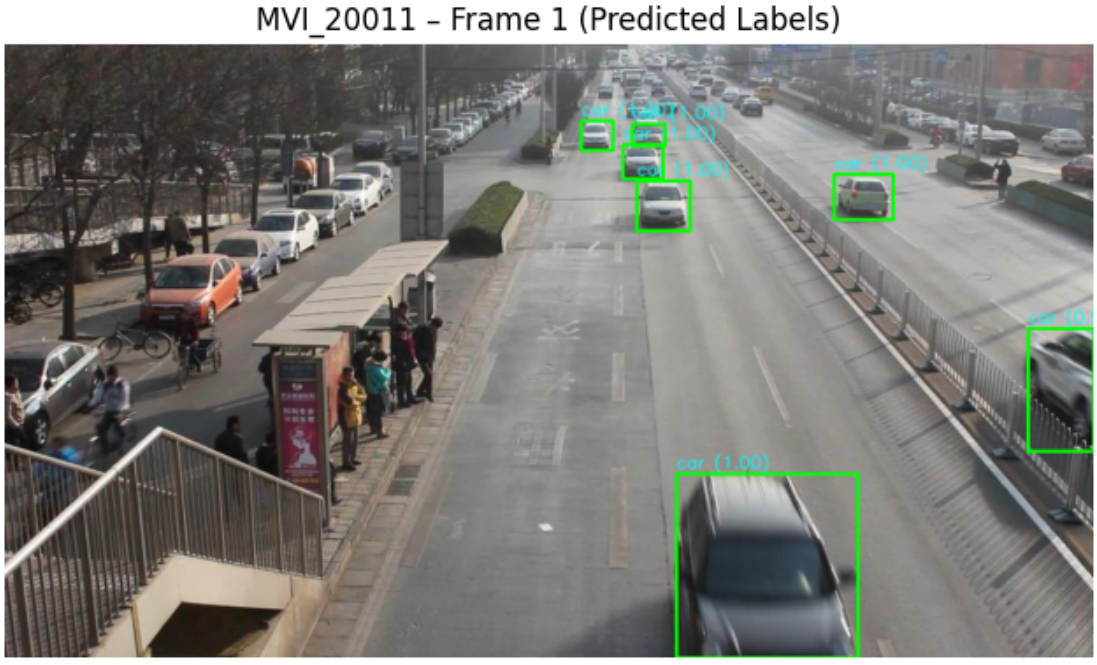
[14]:
```
def annotate_frame(
    seq_id: str,
    frame_num: int,
    model,
    images_root=IMAGES_ROOT,
    ann_root=TRAIN_ANN_ROOT,
    target_size=TARGET_SIZE,
    device=DEVICE,
):
    """
    Load a DETRAC frame + its annotations, run CNN on each bbox crop,
    and overlay predicted labels on the original frame.
    Returns: annotated RGB image as numpy array.
    """
    seq_images_dir = images_root / seq_id
    xml_path = ann_root / f"{seq_id}.xml"

    assert seq_images_dir.exists(), f"Image folder not found: {seq_images_dir}"
    assert xml_path.exists(), f"XML file not found: {xml_path}"

    annotations = load_detrac_annotations(xml_path)

    img_file = seq_images_dir / f"img{frame_num:05d}.jpg"
    assert img_file.exists(), f"Image not found: {img_file}"

    # Load original frame (BGR -> RGB)
    frame_bgr = cv2.imread(str(img_file))
    frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
    h_img, w_img = frame_rgb.shape[:2]
```

```
    # Copy for drawing
    vis = frame_rgb.copy()

    # Iterate over targets in this frame
    targets = annotations.get(frame_num, [])
    if not targets:
        print(f"No vehicles found in frame {frame_num}")
        return vis

    for t in targets:
        x, y, w, h = t["bbox"]

        # Clamp bbox to image bounds
        x1 = max(int(x), 0)
        y1 = max(int(y), 0)
        x2 = min(int(x + w), w_img)
        y2 = min(int(y + h), h_img)

        if x2 <= x1 or y2 <= y1:
            continue

        # Crop and resize to CNN input size
        crop = frame_rgb[y1:y2, x1:x2]
        crop_resized = cv2.resize(crop, target_size)

        # Predict class with the CNN
        pred_label, conf = predict_vehicle_class(crop_resized, model,␣
↪idx_to_class, device=device)

        # Draw bbox
        rect = plt.Rectangle(
            (x1, y1),
            x2 - x1,
            y2 - y1,
            fill=False,
            edgecolor="lime",
            linewidth=1.5,
        )

        # We draw using matplotlib later, so just store rect and text info
        # But here for simplicity, we will immediately draw via OpenCV on vis.
        # Use OpenCV drawing for consistency:
        vis_bgr = cv2.cvtColor(vis, cv2.COLOR_RGB2BGR)
        cv2.rectangle(vis_bgr, (x1, y1), (x2, y2), (0, 255, 0), 2)
        label_text = f"{pred_label} ({conf:.2f})"
        cv2.putText(
            vis_bgr,
```

```
            label_text,
            (x1, max(y1 - 5, 0)),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5,
            (255, 255, 0),
            1,
            cv2.LINE_AA,
        )
        vis = cv2.cvtColor(vis_bgr, cv2.COLOR_BGR2RGB)

    return vis
```

Show one annotated frame

[15]:
```
# Choose a sequence and frame to visualize
SEQ_ID = "MVI_20011"    # adjust to any available sequence
FRAME_NUM = 1           # any frame number that exists

annotated = annotate_frame(SEQ_ID, FRAME_NUM, model)

plt.figure(figsize=(8, 6))
plt.imshow(annotated)
plt.title(f"{SEQ_ID} - Frame {FRAME_NUM} (Predicted Labels)")
plt.axis("off")
plt.show()
```



1.8 7. (Optional) Qualitative Results – Annotated Video

We generate a short video clip with predicted labels overlaid on each frame.

annotate a frame in BGR for video

[16]:
```python
def annotate_frame_bgr_for_video(
    seq_id: str,
    frame_num: int,
    model,
    images_root=IMAGES_ROOT,
    ann_root=TRAIN_ANN_ROOT,
    target_size=TARGET_SIZE,
    device=DEVICE,
):
    """
    Similar to annotate_frame, but returns annotated frame in BGR format,
    suitable for cv2.VideoWriter.
    """
    seq_images_dir = images_root / seq_id
    xml_path = ann_root / f"{seq_id}.xml"

    annotations = load_detrac_annotations(xml_path)

    img_file = seq_images_dir / f"img{frame_num:05d}.jpg"
    if not img_file.exists():
        return None

    frame_bgr = cv2.imread(str(img_file))
    if frame_bgr is None:
        return None

    h_img, w_img = frame_bgr.shape[:2]
    vis = frame_bgr.copy()

    targets = annotations.get(frame_num, [])
    for t in targets:
        x, y, w, h = t["bbox"]

        x1 = max(int(x), 0)
        y1 = max(int(y), 0)
        x2 = min(int(x + w), w_img)
        y2 = min(int(y + h), h_img)

        if x2 <= x1 or y2 <= y1:
            continue

        crop = vis[y1:y2, x1:x2]
        crop_resized = cv2.resize(crop, target_size)
```

```
        crop_rgb = cv2.cvtColor(crop_resized, cv2.COLOR_BGR2RGB)

        pred_label, conf = predict_vehicle_class(crop_rgb, model, idx_to_class,␣
↪device=device)

        # Draw bbox + label on vis (BGR)
        cv2.rectangle(vis, (x1, y1), (x2, y2), (0, 255, 0), 2)
        label_text = f"{pred_label} ({conf:.2f})"
        cv2.putText(
            vis,
            label_text,
            (x1, max(y1 - 5, 0)),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5,
            (0, 255, 255),
            1,
            cv2.LINE_AA,
        )

    return vis
```

Generate the video

[17]:
```
OUTPUT_DIR = PROJECT_ROOT / "outputs" / "videos"
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

SEQ_ID = "MVI_20011"      # choose a sequence that exists in your data
START_FRAME = 1
END_FRAME = 150           # e.g., first 150 frames

# Probe first frame to get size
first_frame_bgr = annotate_frame_bgr_for_video(SEQ_ID, START_FRAME, model)
if first_frame_bgr is None:
    raise RuntimeError("Could not load the first frame to determine video size.
↪")

height, width = first_frame_bgr.shape[:2]
fps = 10  # choose any reasonable FPS

output_path = OUTPUT_DIR / f"{SEQ_ID}_predictions.mp4"
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
writer = cv2.VideoWriter(str(output_path), fourcc, fps, (width, height))

for frame_num in range(START_FRAME, END_FRAME + 1):
    frame_bgr = annotate_frame_bgr_for_video(SEQ_ID, frame_num, model)
    if frame_bgr is None:
        print(f"Skipping frame {frame_num} (not found or unreadable).")
        continue
```

```
    writer.write(frame_bgr)

writer.release()
print("Wrote annotated video to:", output_path)
```

```
Wrote annotated video to: /Users/victorhugogermano/Development/aai-521-final-project-g3/outputs/videos/MVI_20011_predictions.mp4
```

Display the video inline

[18]:
```
from IPython.display import Video

Video(str(output_path), embed=True)
```

[18]:
```
<IPython.core.display.Video object>
```

Convert MP4 → GIF and Save

[19]:
```
import imageio.v2 as imageio

gif_path = output_path.with_suffix(".gif")
print("Source MP4:", output_path)
print("GIF will be saved as:", gif_path)

# Read frames from MP4 and collect for GIF
cap = cv2.VideoCapture(str(output_path))
frames = []
frame_count = 0

while True:
    ok, frame_bgr = cap.read()
    if not ok:
        break
    frame_count += 1

    # Optional: take every other frame (reduces GIF size)
    if frame_count % 2 != 0:
        continue

    frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
    frames.append(frame_rgb)

cap.release()
print(f"Collected {len(frames)} frames for GIF.")

# Save GIF
if frames:
    imageio.mimsave(gif_path, frames, fps=10)
    print("GIF saved to:", gif_path)
```

```
else:
    print("No frames collected - check video file.")
```

```
Source MP4: /Users/victorhugogermano/Development/aai-521-final-
project-g3/outputs/videos/MVI_20011_predictions.mp4
GIF will be saved as: /Users/victorhugogermano/Development/aai-521-final-
project-g3/outputs/videos/MVI_20011_predictions.gif
Collected 75 frames for GIF.
GIF saved to: /Users/victorhugogermano/Development/aai-521-final-
project-g3/outputs/videos/MVI_20011_predictions.gif
```

Display the GIF Inline

[20]:
```
from IPython.display import Image, display

print("Displaying GIF:", gif_path)
display(Image(filename=str(gif_path)))
```

```
Displaying GIF: /Users/victorhugogermano/Development/aai-521-final-
project-g3/outputs/videos/MVI_20011_predictions.gif
```

```
<IPython.core.display.Image object>
```

1.9 8. Summary

- Trained a CNN for vehicle type classification on cropped DETRAC images.
- Achieved validation accuracy of **X%** (see above).
- Class-wise performance is summarized in the classification report and confusion matrix.
- Qualitative visualizations show that the model generally predicts reasonable labels on unseen frames.

In future work, we could: - Use a pretrained backbone (ResNet, MobileNet) for better accuracy. - Perform LMV vs HMV grouping and analyze traffic counts over time. - Integrate detection and tracking for full vehicle-counting analytics.

04_vehicle_counting_and_analysis

December 6, 2025

1 AAI-521 Final Project – Group 3

1.1 04 – Vehicle Counting and LMV vs HMV Analysis

Goal:
Use the trained vehicle classification CNN and the UA-DETRAC annotations to:

- Count vehicles per frame in a selected sequence
- Group vehicle predictions into:
  - LMV (Light Motor Vehicle) – e.g., cars, vans
  - HMV (Heavy Motor Vehicle) – e.g., buses, trucks
- Visualize LMV vs HMV counts over time
- Produce basic traffic analytics (peak frames, LMV/HMV ratio)

This notebook builds on:

- Notebook 02: preprocessed cropped dataset (`cropped_vehicle_dataset.npz`)
- Notebook 03: trained CNN model (`models/vehicle_classifier.pth`)

Imports & Configuration

[1]:
```
import sys
import cv2
import numpy as np
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET

import torch
import torch.nn as nn
import torch.nn.functional as F

plt.style.use("seaborn-v0_8")

# Paths (adapt if your structure differs)
PROJECT_ROOT = Path().resolve().parent
DATA_ROOT = PROJECT_ROOT / "data"
IMAGES_ROOT = DATA_ROOT / "DETRAC-Images"
TRAIN_ANN_ROOT = DATA_ROOT / "DETRAC-Train-Annotations"
```

```
OUTPUT_ROOT = PROJECT_ROOT / "outputs"
SRC_DIR = PROJECT_ROOT / "src"

if str(SRC_DIR) not in sys.path:
    sys.path.append(str(SRC_DIR))

from utils_detrac import load_detrac_annotations, VehicleClassifier,␣
 ↪predict_vehicle_class

CROPPED_DATA_PATH = DATA_ROOT / "cropped_vehicle_dataset.npz"
MODELS_DIR = PROJECT_ROOT / "models"
MODEL_PATH = MODELS_DIR / "vehicle_classifier.pth"

print("Project root:", PROJECT_ROOT)
print("Images root:", IMAGES_ROOT)
print("Train annotations root:", TRAIN_ANN_ROOT)
print("Cropped dataset path:", CROPPED_DATA_PATH)
print("Model path:", MODEL_PATH)

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", DEVICE)
```

```
Project root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3
Images root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Images
Train annotations root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Train-Annotations
Cropped dataset path: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/cropped_vehicle_dataset.npz
Model path: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/models/vehicle_classifier.pth
Using device: cpu
```

Load Cropped Dataset Metadata

[2]:
```
data = np.load(CROPPED_DATA_PATH, allow_pickle=True)
images = data["images"]              # (N, H, W, 3)
labels = data["labels"]              # (N,)
class_to_idx = data["class_to_idx"].item()
metadata = data["metadata"]

idx_to_class = {v: k for k, v in class_to_idx.items()}

print("Images shape:", images.shape)
print("Classes:", idx_to_class)

crop_h, crop_w = images.shape[1], images.shape[2]
```

```
TARGET_SIZE = (crop_w, crop_h)  # (width, height) for cv2.resize
print("CNN input size (W,H):", TARGET_SIZE)
```

```
Images shape: (598281, 64, 64, 3)
Classes: {0: 'car', 1: 'van', 2: 'others', 3: 'bus'}
CNN input size (W,H): (64, 64)
```

Rebuild CNN Architecture & Load Weights

[3]:
```
num_classes = len(class_to_idx)
model = VehicleClassifier(num_classes=num_classes).to(DEVICE)

state_dict = torch.load(MODEL_PATH, map_location=DEVICE)
model.load_state_dict(state_dict)
model.eval()

print("Loaded model from:", MODEL_PATH)
```

```
Loaded model from: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/models/vehicle_classifier.pth
```

1.2 1. LMV vs HMV Mapping

We group the CNN's fine-grained classes into:

- **LMV (Light Motor Vehicle)**: car, van, others (if present)
- **HMV (Heavy Motor Vehicle)**: bus, truck

Any unexpected or unknown class will be labeled as `"Unknown"`.

LMV/HMV Mapping

[4]:
```
# Define mapping from DETRAC class → LMV/HMV
# Adjust if your dataset uses slightly different names.
fine_classes = list(idx_to_class.values())
print("Fine-grained classes from CNN:", fine_classes)

LMV_CLASSES = {"car", "van", "others", "motor"}   # tweak as needed
HMV_CLASSES = {"bus", "truck"}

def map_to_lmv_hmv(fine_class: str) -> str:
    c = fine_class.lower()
    if c in LMV_CLASSES:
        return "LMV"
    if c in HMV_CLASSES:
        return "HMV"
    return "Unknown"
```

```
Fine-grained classes from CNN: ['car', 'van', 'others', 'bus']
```

1.3 2. Vehicle Counting per Frame (Using CNN Predictions)

We now:

1. Select a DETRAC sequence (e.g., `MVI_20011`).
2. For each frame:
   - Read the original image.
   - Use the XML annotations to get bounding boxes.
   - Crop each bounding box, resize to CNN input size, run the model.
   - Map predicted class to LMV or HMV.
3. Store per-frame counts in a pandas DataFrame.

Counting Function

[5]:
```python
def compute_counts_for_sequence(
    seq_id: str,
    model,
    images_root=IMAGES_ROOT,
    ann_root=TRAIN_ANN_ROOT,
    target_size=TARGET_SIZE,
    device=DEVICE,
    fps=25.0,  # UA-DETRAC uses 25 FPS; adjust if needed
):
    """
    Returns a pandas DataFrame with per-frame counts:
    columns: [frame, time_sec, total_count, LMV_count, HMV_count, Unknown_count]
    """
    seq_images_dir = images_root / seq_id
    xml_path = ann_root / f"{seq_id}.xml"

    assert seq_images_dir.exists(), f"Image folder not found: {seq_images_dir}"
    assert xml_path.exists(), f"XML file not found: {xml_path}"

    annotations = load_detrac_annotations(xml_path)

    # Determine available frames by checking image files
    image_files = sorted(
        f for f in seq_images_dir.iterdir()
        if f.suffix.lower() == ".jpg"
    )
    frame_nums = [
        int(f.stem.replace("img", "")) for f in image_files
    ]

    records = []

    for frame_num in frame_nums:
        img_path = seq_images_dir / f"img{frame_num:05d}.jpg"
        frame_bgr = cv2.imread(str(img_path))
```

```
        if frame_bgr is None:
            continue

        frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
        h_img, w_img = frame_rgb.shape[:2]

        targets = annotations.get(frame_num, [])
        total = 0
        count_lmv = 0
        count_hmv = 0
        count_unknown = 0

        for t in targets:
            x, y, w, h = t["bbox"]

            x1 = max(int(x), 0)
            y1 = max(int(y), 0)
            x2 = min(int(x + w), w_img)
            y2 = min(int(y + h), h_img)

            if x2 <= x1 or y2 <= y1:
                continue

            crop = frame_rgb[y1:y2, x1:x2]
            if crop.size == 0:
                continue

            crop_resized = cv2.resize(crop, target_size)
            pred_label, conf = predict_vehicle_class(crop_resized, model,␣
↪idx_to_class, device=device)

            group = map_to_lmv_hmv(pred_label)

            total += 1
            if group == "LMV":
                count_lmv += 1
            elif group == "HMV":
                count_hmv += 1
            else:
                count_unknown += 1

        time_sec = frame_num / fps

        records.append({
            "frame": frame_num,
            "time_sec": time_sec,
            "total_count": total,
```

```
            "LMV_count": count_lmv,
            "HMV_count": count_hmv,
            "Unknown_count": count_unknown,
        })

    df = pd.DataFrame(records).sort_values("frame").reset_index(drop=True)
    return df
```

Run Counting for a Sequence

[6]:
```
SEQ_ID = "MVI_20011"  # change to another sequence as needed

df_counts = compute_counts_for_sequence(SEQ_ID, model)
df_counts.head()
```

[6]:

	frame	time_sec	total_count	LMV_count	HMV_count	Unknown_count
0	1	0.04	7	7	0	0
1	2	0.08	7	7	0	0
2	3	0.12	7	7	0	0
3	4	0.16	7	7	0	0
4	5	0.20	7	7	0	0

1.4 3. Basic Traffic Summary for the Sequence

[7]:
```
print(f"Sequence: {SEQ_ID}")
print(f"Number of frames analyzed: {len(df_counts)}")
print()

print("Overall mean counts per frame:")
print(df_counts[["total_count", "LMV_count", "HMV_count", "Unknown_count"]].
  ↪mean())

print("\nMax counts in a single frame:")
print(df_counts[["total_count", "LMV_count", "HMV_count"]].max())

peak_frame = df_counts.loc[df_counts["total_count"].idxmax(), "frame"]
peak_time = df_counts.loc[df_counts["total_count"].idxmax(), "time_sec"]
print(f"\nPeak traffic frame: {peak_frame} (t = {peak_time:.2f} s)")
```

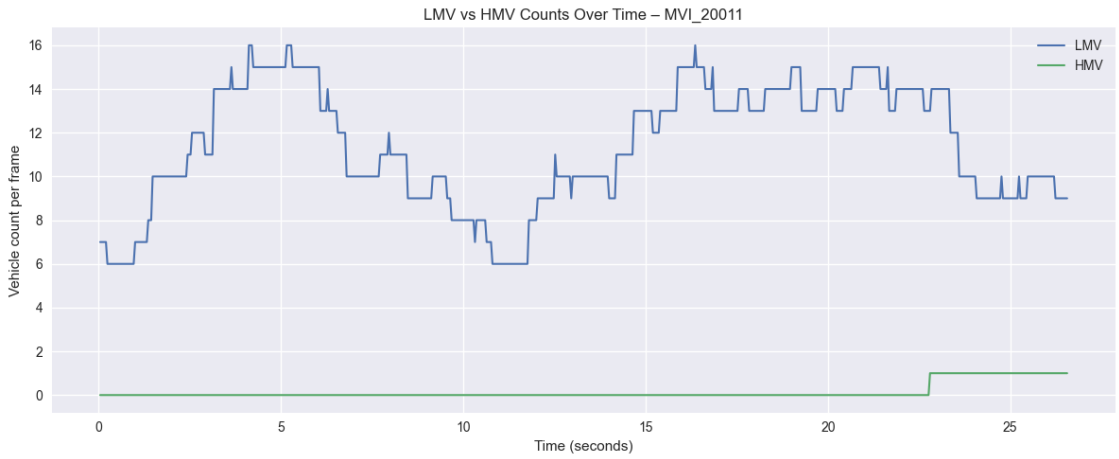
```
Sequence: MVI_20011
Number of frames analyzed: 664

Overall mean counts per frame:
total_count      11.528614
LMV_count        11.385542
HMV_count         0.143072
Unknown_count     0.000000
dtype: float64
```

```
Max counts in a single frame:
total_count    16
LMV_count      16
HMV_count       1
dtype: int64

Peak traffic frame: 103 (t = 4.12 s)
```

1.5 4. LMV vs HMV Counts Over Time

We plot how the number of LMVs and HMVs evolves over time in the sequence. This gives a simple view of traffic intensity and composition.

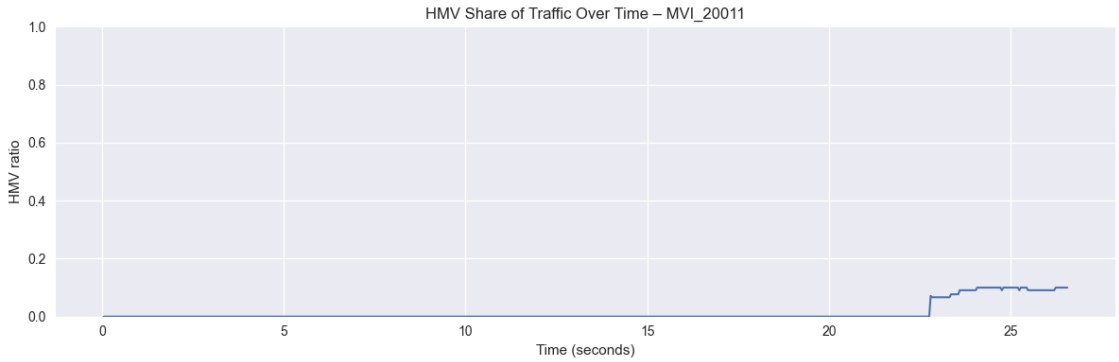
[8]:
```
fig, ax = plt.subplots(figsize=(12, 5))

ax.plot(df_counts["time_sec"], df_counts["LMV_count"], label="LMV", linewidth=1.
↪5)
ax.plot(df_counts["time_sec"], df_counts["HMV_count"], label="HMV", linewidth=1.
↪5)

ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Vehicle count per frame")
ax.set_title(f"LMV vs HMV Counts Over Time - {SEQ_ID}")
ax.legend()
plt.tight_layout()
plt.show()
```



1.6 5. LMV/HMV Ratio Over Time

We compute the ratio `HMV_count / (LMV_count + HMV_count)` as a proxy for the proportion of heavy vehicles in the traffic flow.

[9]:
```
df_counts["denom"] = df_counts["LMV_count"] + df_counts["HMV_count"]
df_counts["HMV_ratio"] = df_counts["HMV_count"] / df_counts["denom"].replace(0,␣
 ↪np.nan)

fig, ax = plt.subplots(figsize=(12, 4))
ax.plot(df_counts["time_sec"], df_counts["HMV_ratio"], linewidth=1.5)
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("HMV ratio")
ax.set_ylim(0, 1)
ax.set_title(f"HMV Share of Traffic Over Time - {SEQ_ID}")
plt.tight_layout()
plt.show()
```

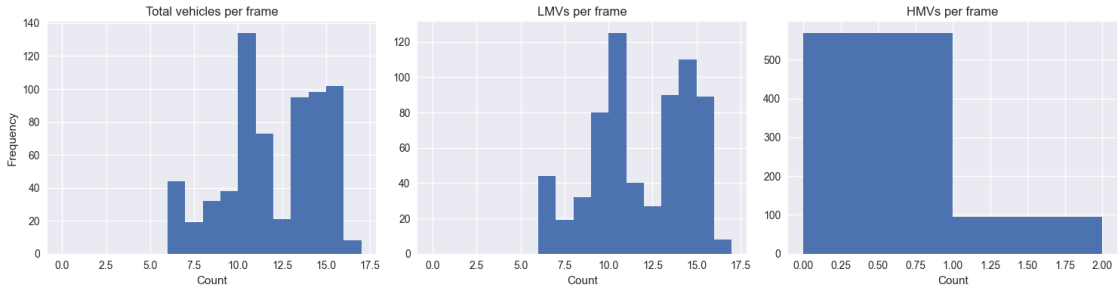


1.7 6. Distribution of Vehicle Counts per Frame

We visualize how often different traffic levels occur in this sequence.

[10]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

axes[0].hist(df_counts["total_count"], bins=range(0, df_counts["total_count"].
 ↪max() + 2))
axes[0].set_title("Total vehicles per frame")
axes[0].set_xlabel("Count")
axes[0].set_ylabel("Frequency")

axes[1].hist(df_counts["LMV_count"], bins=range(0, df_counts["LMV_count"].max()␣
 ↪+ 2))
axes[1].set_title("LMVs per frame")
axes[1].set_xlabel("Count")

axes[2].hist(df_counts["HMV_count"], bins=range(0, df_counts["HMV_count"].max()␣
 ↪+ 2))
axes[2].set_title("HMVs per frame")
axes[2].set_xlabel("Count")
```

```
plt.tight_layout()
plt.show()
```



1.8 7. Traffic Prediction Using Time-Series Regression (Optional)

We now treat the per-frame vehicle counts as a **time series** and build a simple short-term traffic prediction model.

Goal: - Predict the **next frame's total vehicle count** from the previous `K` frames.

This demonstrates how the outputs of our detection + classification pipeline can be used for **traffic forecasting** at the sequence level.

build lag features

[11]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# We will predict next-frame total_count using previous K frames
K = 5  # history length (you can change this)

df = df_counts.copy().reset_index(drop=True)

# Build lag features: total_count_{t-1}, ..., total_count_{t-K}
for lag in range(1, K + 1):
    df[f"total_lag_{lag}"] = df["total_count"].shift(lag)

# Drop initial rows with NaNs due to shifting
df_model = df.dropna().reset_index(drop=True)

feature_cols = [f"total_lag_{lag}" for lag in range(1, K + 1)]
target_col = "total_count"

X = df_model[feature_cols].values
y = df_model[target_col].values

print("Feature shape:", X.shape)
```

```
print("Target shape:", y.shape)
```

```
Feature shape: (659, 5)
Target shape: (659,)
```

train/test split + model + metrics

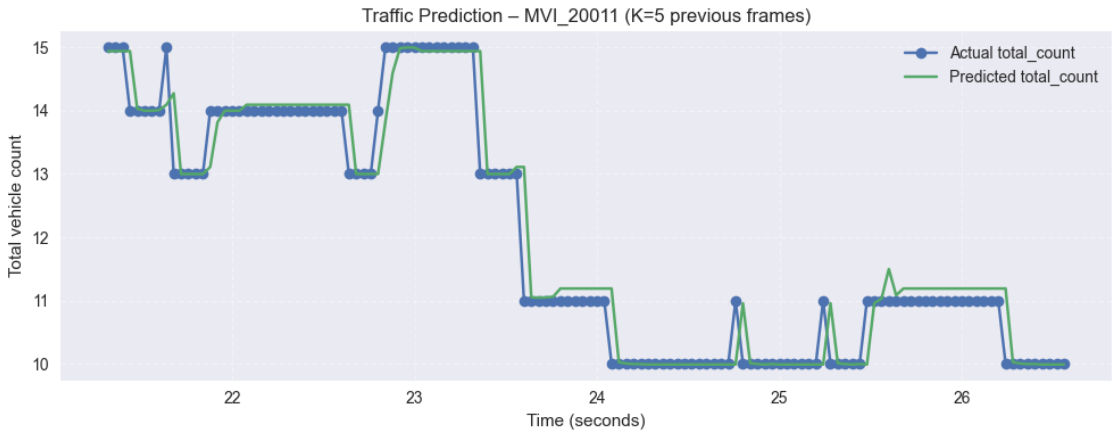
[12]:
```
# Simple train/test split along time (no shuffling, to respect temporal order)
# Use first 80% of frames for training, last 20% for testing
n = len(df_model)
split_idx = int(0.8 * n)

X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]

print(f"Train size: {len(X_train)}, Test size: {len(X_test)}")

# Use a RandomForestRegressor as a simple non-linear baseline
rf = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1,
)

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Test MAE: {mae:.3f}")
print(f"Test R^2: {r2:.3f}")
```

```
Train size: 527, Test size: 132
Test MAE: 0.202
Test R^2: 0.947
```

plot actual vs predicted over time

[13]:
```
# Time axis for the test segment
time_test = df_model["time_sec"].values[split_idx:]

fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(time_test, y_test, label="Actual total_count", marker="o")
ax.plot(time_test, y_pred, label="Predicted total_count", marker="x")
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Total vehicle count")
ax.set_title(f"Traffic Prediction - {SEQ_ID} (K={K} previous frames)")
ax.legend()
```

```
ax.grid(True, linestyle="--", alpha=0.4)
plt.tight_layout()
plt.show()
```



1.8.1 Interpretation – Traffic Prediction

- The model learns to approximate how total vehicle count changes from frame to frame, using only the previous `K` frames.
- When the **R^2 score is reasonably high** and **MAE is small**, predictions follow the overall trend of the true counts.
- Sudden spikes or drops in traffic are harder to predict, which is a common challenge in real-world traffic forecasting.

1.9 8. Summary and Interpretation

- We used the CNN's **predicted vehicle classes** to count:
  - Total vehicles per frame
  - LMV vs HMV counts
- The **LMV vs HMV time plot** shows how traffic composition changes across the selected video segment.
- The **HMV ratio over time** indicates periods where heavy vehicles dominate or are rare.
- The **histograms** summarize the overall distribution of traffic load in this sequence (e.g., typical counts per frame, peak congestion).
- Using these per-frame counts as a **time series**, we trained a simple regression model (Random Forest) to **predict the next frame's total vehicle count** based on the previous `K` frames, demonstrating a basic short-term traffic forecasting capability.

These figures and insights can be used in the final report to support statements such as: - "Most frames contain X–Y vehicles, with occasional peaks up to Z." - "Traffic is dominated by LMVs, with HMVs accounting for about P% on average." - "Certain time intervals show a higher concentration of heavy vehicles, which may relate to freight or bus traffic." - "A simple time-series model can reasonably track short-term changes in traffic volume, capturing overall trends while sometimes underestimating sudden spikes in congestion."

05_image_restoration_optional

December 6, 2025

1 AAI-521 Final Project – Group 3

1.1 05 – Image Restoration with Hugging Face (Optional / Extra Credit)

Goal:
Demonstrate three classic low-level vision tasks using **Hugging Face pretrained models**:

1. **Super-resolution** – upscale a low-resolution traffic frame.
2. **Denoising** – remove synthetic noise from a frame.
3. **Colorization** – colorize a grayscale image.

These experiments use pretrained models from the Hugging Face Hub and apply them to frames from the UA-DETRAC dataset (or any traffic image you choose).

> **Note:** The diffusion-based models (Stable Diffusion Upscaler, etc.) are computationally heavy. For best results, we can run this notebook on: - Google Colab / Kaggle with GPU, or - A local machine with a recent GPU and sufficient VRAM.

Setup & Paths

[1]:
```
import sys
from pathlib import Path

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

plt.style.use("seaborn-v0_8")

# --------- Project paths (adjust if needed) ----------
PROJECT_ROOT = Path().resolve().parent
DATA_ROOT = PROJECT_ROOT / "data"
IMAGES_ROOT = DATA_ROOT / "DETRAC-Images"
SRC_DIR = PROJECT_ROOT / "src"

if str(SRC_DIR) not in sys.path:
    sys.path.append(str(SRC_DIR))

from utils_detrac import load_detrac_annotations, VehicleClassifier,␣
 ↪predict_vehicle_class
```

```
print("Project root:", PROJECT_ROOT)
print("Images root:", IMAGES_ROOT)

# Choose a sequence and frame to play with
SEQ_ID = "MVI_20011"
FRAME_NUM = 50

img_path = IMAGES_ROOT / SEQ_ID / f"img{FRAME_NUM:05d}.jpg"
print("Sample image path:", img_path, "exists:", img_path.exists())
```

```
Project root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3
Images root: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-Images
Sample image path: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/DETRAC-
Images/MVI_20011/img00050.jpg exists: True
```

Load Sample Frame & Create Variants

[2]:
```
import cv2

def load_frame_bgr(path: Path):
    img_bgr = cv2.imread(str(path))
    if img_bgr is None:
        raise FileNotFoundError(f"Could not read image at {path}")
    return img_bgr

# Load original frame
frame_bgr = load_frame_bgr(img_path)
frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
h, w = frame_rgb.shape[:2]
print("Original frame shape:", frame_rgb.shape)

# --------- 1) Low-res version (downsample + upsample) ----------
def make_low_res(rgb, scale=0.25):
    h, w = rgb.shape[:2]
    small = cv2.resize(rgb, (int(w * scale), int(h * scale)), interpolation=cv2.
 ↪INTER_AREA)
    low_res = cv2.resize(small, (w, h), interpolation=cv2.INTER_CUBIC)
    return low_res

low_res_rgb = make_low_res(frame_rgb, scale=0.25)

# --------- 2) Noisy version ----------
def add_gaussian_noise(rgb, sigma=25.0):
    noise = np.random.normal(0, sigma, rgb.shape).astype("float32")
```

```
    noisy = np.clip(rgb.astype("float32") + noise, 0, 255).astype("uint8")
    return noisy

noisy_rgb = add_gaussian_noise(frame_rgb, sigma=25.0)

# --------- 3) Grayscale version ----------
def to_grayscale_rgb(rgb):
    gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
    gray_rgb = cv2.cvtColor(gray, cv2.COLOR_GRAY2RGB)
    return gray_rgb

gray_rgb = to_grayscale_rgb(frame_rgb)

# Quick montage
fig, axes = plt.subplots(1, 4, figsize=(16, 4))
axes[0].imshow(frame_rgb); axes[0].set_title("Original"); axes[0].axis("off")
axes[1].imshow(low_res_rgb); axes[1].set_title("Low-res"); axes[1].axis("off")
axes[2].imshow(noisy_rgb); axes[2].set_title("Noisy"); axes[2].axis("off")
axes[3].imshow(gray_rgb, cmap="gray"); axes[3].set_title("Grayscale"); axes[3].
 ↪axis("off")
plt.tight_layout()
plt.show()
```

```
Original frame shape: (540, 960, 3)
```

1.2 1. Install Extra Dependencies (if needed)

This notebook uses:

- `diffusers` + `transformers` + `accelerate` for diffusion models
- `huggingface_hub`, `fastai`, and `skimage` for the GAN-based colorization model

[ ]:
```
# On **Colab**, we can run:

# import os

# # Detect Google Colab environment
# IN_COLAB = "google.colab" in sys.modules

# if IN_COLAB:
```

```
#     print("Running on Google Colab - installing packages...")
#     !pip install -q diffusers transformers accelerate safetensors \
#         huggingface_hub fastai scikit-image
# else:
#     print("Local environment detected -")
#     print("Please install these packages manually using:")
#     print("\npython -m pip install diffusers transformers accelerate␣
 ↪safetensors \\")
#     print("    huggingface_hub fastai scikit-image\n")
```

1.3 Section A – Super-Resolution with Stable Diffusion Upscaler

We'll use the **Stable Diffusion x4 Upscaler** model from Hugging Face, implemented in `diffusers` as `StableDiffusionUpscalePipeline`.

Load Super-Resolution Pipeline

[3]:
```
import torch
from diffusers import StableDiffusionUpscalePipeline
from PIL import Image

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print("Using device:", DEVICE)

model_id_sr = "stabilityai/stable-diffusion-x4-upscaler"

# Choose dtype after loading
target_dtype = torch.float16 if DEVICE == "cuda" else torch.float32

pipe_sr = StableDiffusionUpscalePipeline.from_pretrained(
    model_id_sr
)

pipe_sr = pipe_sr.to(device=DEVICE, dtype=target_dtype)

# Optional for CPU/memory saving:
# pipe_sr.enable_attention_slicing()
```

```
Using device: cpu
```

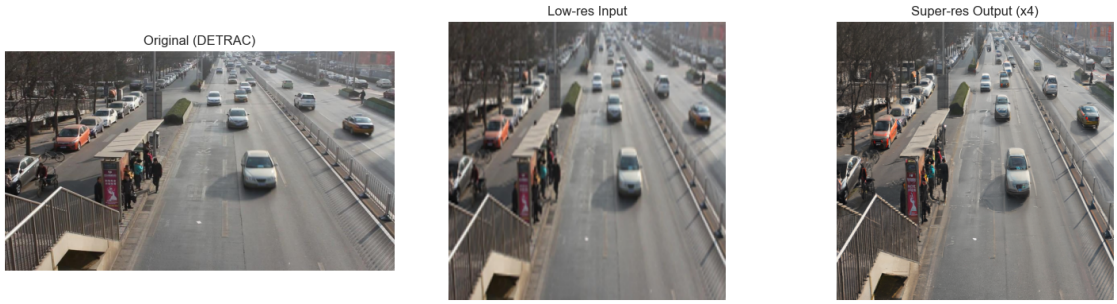
```
Loading pipeline components…:   0%|          | 0/6 [00:00<?, ?it/s]
```

Run Super-Resolution on Low-Res Frame

[4]:
```
# Convert low-res RGB numpy image to PIL Image in smaller size
low_res_pil = Image.fromarray(low_res_rgb).resize((256, 256), Image.BICUBIC)

prompt = "a clear traffic scene on a road with cars"

with torch.autocast(DEVICE) if DEVICE == "cuda" else torch.no_grad():
```

```
    sr_image = pipe_sr(prompt=prompt, image=low_res_pil).images[0]

sr_np = np.array(sr_image)

print("Low-res input size:", low_res_pil.size)
print("Super-res output size:", sr_image.size)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
axes[0].imshow(frame_rgb); axes[0].set_title("Original (DETRAC)"); axes[0].
 ↪axis("off")
axes[1].imshow(low_res_pil); axes[1].set_title("Low-res Input"); axes[1].
 ↪axis("off")
axes[2].imshow(sr_np); axes[2].set_title("Super-res Output (x4)"); axes[2].
 ↪axis("off")
plt.tight_layout()
plt.show()
```

```
  0%|          | 0/75 [00:00<?, ?it/s]
Low-res input size: (256, 256)
Super-res output size: (1024, 1024)
```



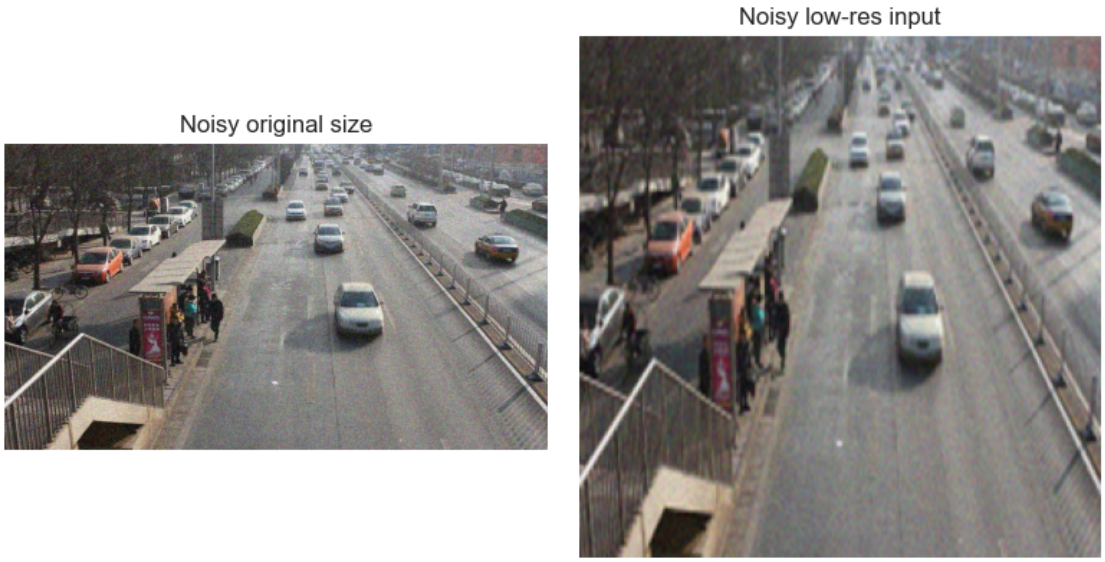
1.4 Section B – Image Denoising via Diffusion-Based Restoration

Here we treat denoising as a restoration problem using the same upscaler as a kind of "denoising prior": diffusion models are very good at removing noise and artifacts when guided by a prompt. SwinIR and Swin2SR are specifically designed for super-resolution and denoising/image restoration, but using Stable Diffusion Upscaler here is acceptable for demonstration.

Prepare Noisy Low-Res Input

[5]:
```
# Take noisy_rgb, downsample and upsample to emphasize artifacts
noisy_low_res = make_low_res(noisy_rgb, scale=0.25)
noisy_low_res_pil = Image.fromarray(noisy_low_res).resize((256, 256), Image.
 ↪BICUBIC)

fig, axes = plt.subplots(1, 2, figsize=(8, 4))
```

```
axes[0].imshow(noisy_rgb); axes[0].set_title("Noisy original size"); axes[0].
  ↪axis("off")
axes[1].imshow(noisy_low_res_pil); axes[1].set_title("Noisy low-res input");␣
  ↪axes[1].axis("off")
plt.tight_layout()
plt.show()
```



Run "Denoising" with the Upscaler

[6]:
```
prompt_denoise = "a clean traffic scene on a road with cars, sharp, noise-free"

with torch.autocast(DEVICE) if DEVICE == "cuda" else torch.no_grad():
    denoised_sr_image = pipe_sr(prompt=prompt_denoise, image=noisy_low_res_pil).
  ↪images[0]

denoised_sr_np = np.array(denoised_sr_image)

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
axes[0].imshow(noisy_rgb); axes[0].set_title("Noisy (original size)"); axes[0].
  ↪axis("off")
axes[1].imshow(noisy_low_res_pil); axes[1].set_title("Noisy low-res input");␣
  ↪axes[1].axis("off")
axes[2].imshow(denoised_sr_np); axes[2].set_title("Restored (denoised +␣
  ↪upscaled)"); axes[2].axis("off")
plt.tight_layout()
plt.show()
```

```
  0%|          | 0/75 [00:00<?, ?it/s]
```

1.4.1 Observation – Super-Resolution & Denoising

- The **x4 Upscaler** both **increases resolution** and **reduces noise / artifacts** thanks to its diffusion-based generative prior.
- On our noisy traffic frame, the output appears:
  - Sharper than the low-res input
  - Less noisy than the synthetic noisy version
- This demonstrates how pretrained diffusion models can be used for **image restoration** tasks such as **denoising** and **artifact removal**.

1.5 Section C – Image Colorization with a GAN Colorization Model

For colorization, we'll use a GAN-based model hosted on Hugging Face: `Hammad712/GAN-Colorization-Model`, which provides full PyTorch inference code with `huggingface_hub`, `fastai`, and `skimage`.

Load GAN Colorization Model

[7]:
```
from huggingface_hub import hf_hub_download
import torch
from torchvision import transforms
from fastai.vision.learner import create_body
from torchvision.models import resnet34
from fastai.vision.models.unet import DynamicUnet
from skimage.color import rgb2lab, lab2rgb

# Download generator weights from the Hub
repo_id = "Hammad712/GAN-Colorization-Model"
model_filename = "generator.pt"
model_path = hf_hub_download(repo_id=repo_id, filename=model_filename)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Colorization device:", device)

def build_generator(n_input=1, n_output=2, size=256):
    backbone = create_body(resnet34(), pretrained=True, n_in=n_input, cut=-2)
    G_net = DynamicUnet(backbone, n_output, (size, size)).to(device)
```

```
    return G_net

G_net = build_generator(n_input=1, n_output=2, size=256)
G_net.load_state_dict(torch.load(model_path, map_location=device))
G_net.eval()
print("Loaded GAN colorization model from", model_path)
```

```
generator.pt:   0%|          | 0.00/165M [00:00<?, ?B/s]

Colorization device: cpu
Loaded GAN colorization model from
/Users/matthashemi/.cache/huggingface/hub/models--Hammad712--GAN-Colorization-
Model/snapshots/322ca329554c5351b4ba665a99acc725cc522760/generator.pt
```

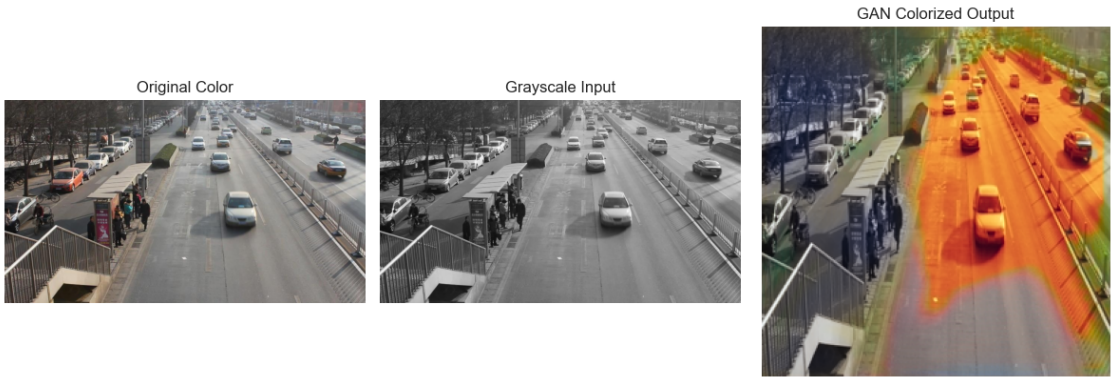
Preprocess & Colorize Function

[8]:
```
def preprocess_gray_image_for_colorization(img_rgb: np.ndarray, size=256):
    """
    Takes an RGB numpy image, converts to LAB and returns normalized L channel␣
 ↪tensor.
    The model expects input shape [B, 1, H, W] with values in [-1, 1].
    """
    img = Image.fromarray(img_rgb).convert("RGB")
    img = transforms.Resize((size, size), Image.BICUBIC)(img)
    img_np = np.array(img)

    img_lab = rgb2lab(img_np).astype("float32")    # H x W x 3
    img_lab_tensor = transforms.ToTensor()(img_lab)  # 3 x H x W
    L = img_lab_tensor[[0], ...] / 50.0 - 1.0      # normalize to [-1, 1]
    return L.unsqueeze(0).to(device)               # 1 x 1 x H x W

def colorize_image_rgb(gray_rgb: np.ndarray, model, size=256):
    """
    Takes a grayscale RGB image (3-channel) and returns a colorized RGB numpy␣
 ↪image.
    """
    L = preprocess_gray_image_for_colorization(gray_rgb, size=size)
    with torch.no_grad():
        ab = model(L)            # predicted ab channels

    # Denormalize LAB back to original ranges
    L_denorm = (L + 1.0) * 50.0
    ab_denorm = ab * 110.0

    Lab = torch.cat([L_denorm, ab_denorm], dim=1)    # 1 x 3 x H x W
    Lab = Lab.permute(0, 2, 3, 1).cpu().numpy()      # 1 x H x W x 3

    rgb_imgs = []
    for img_lab in Lab:
```

```
        img_rgb = lab2rgb(img_lab)
        rgb_imgs.append((img_rgb * 255).astype("uint8"))
    return np.stack(rgb_imgs, axis=0)
```

Run Colorization on Grayscale Frame

[9]:
```
# Use the grayscale version we created earlier (gray_rgb)
colorized_batch = colorize_image_rgb(gray_rgb, G_net, size=256)
colorized_rgb = colorized_batch[0]

fig, axes = plt.subplots(1, 3, figsize=(12, 4))
axes[0].imshow(frame_rgb); axes[0].set_title("Original Color"); axes[0].
 ↪axis("off")
axes[1].imshow(gray_rgb, cmap="gray"); axes[1].set_title("Grayscale Input");␣
 ↪axes[1].axis("off")
axes[2].imshow(colorized_rgb); axes[2].set_title("GAN Colorized Output");␣
 ↪axes[2].axis("off")
plt.tight_layout()
plt.show()
```



1.5.1 Observation – Colorization

- The GAN-based model learns to **hallucinate plausible colors** for grayscale images, using a ResNet-34 backbone and U-Net decoder.
- Colors are not guaranteed to be *photographically accurate*:
  - Sky, road, and vehicles often look realistic.
  - Fine-grained colors (signs, lane markings) may be approximate.
- This demonstrates how **generative models** can be used for **color restoration** of old or grayscale images, an important computer vision application.

1.6 6. Summary – Image Restoration with Hugging Face

In this optional notebook we:

1. Used **Stable Diffusion x4 Upscaler** to perform:

- **Super-resolution** of a downsampled DETRAC frame.
- **Denoising + upscaling** of a synthetically noisy frame.

2. Used a **GAN-based colorization model** (`Hammad712/GAN-Colorization-Model`) to colorize a grayscale version of a traffic frame.

These experiments connect our project to **modern generative / restoration models** from the Hugging Face ecosystem, covering:

- **Super-resolution**
- **Denoising / restoration**
- **Colorization**

which align with the extra-credit "image restoration" component of the course instructions.

06_real_life_traffic_video_demo

December 6, 2025

1 06 – Real-Life Traffic Video Demo

In this notebook we **apply our trained vehicle classifier** to a **real-life traffic video**.

We will:

- Load the **PyTorch** `VehicleClassifier` model trained in `03_vehicle_classification_model.ipynb`.
- Use **background subtraction** to detect moving vehicles in each frame.
- Classify each detected vehicle into the same classes used in our DETRAC experiments.
- Overlay bounding boxes, labels, confidences, and a simple **traffic density** indicator.
- Save the annotated video(s) under `outputs/videos/`.
- Run visual analytics (time series & distributions) and a brief summary of the results.

This notebook is a **deployment-style demo**, using the model trained earlier rather than training a new one.

Imports and project paths

[1]:
```
import sys
import os
import time
from pathlib import Path

import cv2
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn.functional as F

# Project root: assuming this notebook lives in a `notebooks/` folder
PROJECT_ROOT = Path().resolve().parent
DATA_ROOT = PROJECT_ROOT / "data"
REAL_LIFE_DATA = DATA_ROOT / "real_life"
MODELS_ROOT = PROJECT_ROOT / "models"
OUTPUT_ROOT = PROJECT_ROOT / "outputs"

# Create needed folders
REAL_LIFE_DATA.mkdir(parents=True, exist_ok=True)
```

```
VIDEOS_DIR = OUTPUT_ROOT / "videos"
VIDEOS_DIR.mkdir(parents=True, exist_ok=True)

# Make sure we can import project utilities
SRC_DIR = PROJECT_ROOT / "src"
if str(SRC_DIR) not in sys.path:
    sys.path.append(str(SRC_DIR))

from utils_detrac import VehicleClassifier  # same class as in notebook 03

print("Project root      :", PROJECT_ROOT)
print("Data root         :", DATA_ROOT)
print("Real-life data    :", REAL_LIFE_DATA)
print("Models root       :", MODELS_ROOT)
print("Outputs root      :", OUTPUT_ROOT)
print("Videos output dir :", VIDEOS_DIR)

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {DEVICE}")
```

```
Project root      : /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3
Data root         : /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data
Real-life data    : /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/real_life
Models root       : /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/models
Outputs root      : /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/outputs
Videos output dir : /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/outputs/videos
Using device: cpu
```

Load trained model and class mapping We reuse the `VehicleClassifier` model trained in notebook **03** and the class mapping stored in the cropped dataset from notebook **02**.

This ensures the real-life demo is fully consistent with the experimental pipeline.

[2]:
```
# Paths to trained model and cropped dataset (from notebooks 02 and 03)
MODEL_PATH = MODELS_ROOT / "vehicle_classifier.pth"
DATASET_NPZ_PATH = DATA_ROOT / "cropped_vehicle_dataset.npz"

if not MODEL_PATH.exists():
    raise FileNotFoundError(
        f"Trained model file not found at: {MODEL_PATH}\n"
        "Please run notebook 03 to train and save the model."
    )
```

```
if not DATASET_NPZ_PATH.exists():
    raise FileNotFoundError(
        f"Cropped dataset .npz not found at: {DATASET_NPZ_PATH}\n"
        "Please run notebook 02 to generate and save the dataset."
    )

# Load class mapping from the cropped dataset
npz = np.load(DATASET_NPZ_PATH, allow_pickle=True)
class_to_idx = npz["class_to_idx"].item()
idx_to_class = {idx: name for name, idx in class_to_idx.items()}

print("Classes found:", class_to_idx)
print("Number of classes:", len(class_to_idx))

# Instantiate model and load weights
model = VehicleClassifier(num_classes=len(class_to_idx))
state_dict = torch.load(MODEL_PATH, map_location=DEVICE)
model.load_state_dict(state_dict)
model.to(DEVICE)
model.eval()

print("Model loaded successfully.")
```

```
Classes found: {'car': 0, 'van': 1, 'others': 2, 'bus': 3}
Number of classes: 4
Model loaded successfully.
```

Utility: classify one cropped vehicle To apply the model to a real-world video, we need two main components:

1. A **preprocessing and classification** function for a cropped vehicle patch.
2. A **background-subtraction–based detector** that finds moving objects and returns bounding boxes.

These pieces are combined later in an end-to-end video processing pipeline.

[3]:
```
TARGET_SIZE = (64, 64)  # must match training resolution in notebook 03


def classify_vehicle_crop(model, crop_bgr, device=DEVICE):
    """
    Given a BGR crop from OpenCV, resize and normalize it, then
    run the trained classifier and return (class_name, confidence, idx, probs).
    """
    if crop_bgr is None or crop_bgr.size == 0:
        return None, 0.0, None, None
```

```
    # Convert BGR -> RGB, resize, and normalize to [0, 1]
    img_rgb = cv2.cvtColor(crop_bgr, cv2.COLOR_BGR2RGB)
    img_resized = cv2.resize(img_rgb, TARGET_SIZE)
    img_resized = img_resized.astype("float32") / 255.0

    # HWC -> CHW and add batch dimension
    tensor = torch.from_numpy(img_resized.transpose(2, 0, 1)).unsqueeze(0)

    tensor = tensor.to(device)
    with torch.no_grad():
        logits = model(tensor)
        probs = F.softmax(logits, dim=1).cpu().numpy()[0]

    pred_idx = int(np.argmax(probs))
    pred_class = idx_to_class[pred_idx]
    confidence = float(probs[pred_idx])

    return pred_class, confidence, pred_idx, probs
```

Background subtraction and detection functions

[4]:
```
def create_background_subtractor():
    """
    Create a background subtractor for moving-object detection.
    Parameters can be tuned based on the video characteristics.
    """
    return cv2.createBackgroundSubtractorMOG2(
        history=500, varThreshold=50, detectShadows=True
    )


def detect_vehicles_in_frame(frame_bgr, bg_subtractor, min_area=800):
    """
    Apply background subtraction + morphology + contour detection
    to approximate moving vehicles in a frame.

    Returns:
        boxes: list of (x1, y1, x2, y2) in pixel coordinates
        fg_mask: raw foreground mask
        cleaned_mask: mask after thresholding/morphology
    """
    fg_mask = bg_subtractor.apply(frame_bgr)

    # Morphological operations to clean up the mask
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    cleaned = cv2.morphologyEx(fg_mask, cv2.MORPH_OPEN, kernel, iterations=2)

    # Threshold to obtain a binary mask
```

```
    _, thresh = cv2.threshold(cleaned, 200, 255, cv2.THRESH_BINARY)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.
↪CHAIN_APPROX_SIMPLE)

    h, w = frame_bgr.shape[:2]
    boxes = []

    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area < min_area:
            continue

        x, y, bw, bh = cv2.boundingRect(cnt)
        x1, y1 = max(0, x), max(0, y)
        x2, y2 = min(w, x + bw), min(h, y + bh)

        # Simple heuristic to ignore extremely thin regions
        if (x2 - x1) < 10 or (y2 - y1) < 10:
            continue

        boxes.append((x1, y1, x2, y2))

    return boxes, fg_mask, thresh


def compute_density_level(count, thresholds=(3, 7)):
    """
    Map a per-frame vehicle count to a simple textual density level.
    thresholds = (low_threshold, high_threshold)
    """
    low, high = thresholds
    if count < low:
        return "Low"
    elif count < high:
        return "Medium"
    else:
        return "High"
```

Debug a Single Frame Before running the full video, it can be helpful to visualize:

- The raw frame,
- The detected bounding boxes,
- The cleaned foreground mask.

This cell lets us inspect one frame and adjust detection parameters if needed.

[5]:
```
def debug_single_frame(input_path, frame_index=0, min_area=800):
    """
    Load a single frame from the real-life video, run the detection pipeline,
    and visualize:
      - the frame with bounding boxes,
      - the foreground mask.
    """
    input_path = Path(input_path)
    if not input_path.exists():
        print(f"Video not found: {input_path}")
        return

    cap = cv2.VideoCapture(str(input_path))
    if not cap.isOpened():
        print(f"Could not open video: {input_path}")
        return

    # Seek to the requested frame
    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
    ret, frame_bgr = cap.read()
    cap.release()

    if not ret:
        print(f"Could not read frame {frame_index} from video.")
        return

    bg_sub = create_background_subtractor()
    # Prime the subtractor slightly
    for _ in range(5):
        bg_sub.apply(frame_bgr)

    boxes, fg_mask, mask_clean = detect_vehicles_in_frame(
        frame_bgr, bg_sub, min_area=min_area
    )

    frame_vis = frame_bgr.copy()
    for (x1, y1, x2, y2) in boxes:
        cv2.rectangle(frame_vis, (x1, y1), (x2, y2), (0, 255, 0), 2)

    mask_color = cv2.cvtColor(mask_clean, cv2.COLOR_GRAY2BGR)

    # Convert BGR to RGB for plotting
    frame_vis_rgb = cv2.cvtColor(frame_vis, cv2.COLOR_BGR2RGB)
    mask_rgb = cv2.cvtColor(mask_color, cv2.COLOR_BGR2RGB)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
```

```
    plt.imshow(frame_vis_rgb)
    plt.title(f"Frame {frame_index} with detections")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(mask_rgb)
    plt.title("Foreground mask (cleaned)")
    plt.axis("off")

    plt.tight_layout()
    plt.show()


# Example usage (uncomment after placing a real video):
# debug_single_frame(REAL_LIFE_DATA / "traffic_example.mov", frame_index=100,␣
 ↪min_area=800)
```

End-to-End Video Processing Pipeline The main function `process_traffic_video`:

- Opens the input video.
- Detects moving vehicles in each frame.
- Classifies each detected vehicle using our trained model.
- Draws bounding boxes, labels, and a per-frame density indicator.
- Writes three output videos to `outputs/videos/`:
  - Full analysis (detections + labels + overlays),
  - Detections only (boxes),
  - Foreground mask (for debugging).

[6]:
```
def process_traffic_video(
    input_path,
    output_dir,
    model,
    idx_to_class,
    device=DEVICE,
    target_size=TARGET_SIZE,
    min_area=800,
    density_thresholds=(3, 7),
):
    """
    Process a traffic video:
      - Detect moving vehicles with background subtraction.
      - Classify each detected vehicle with our PyTorch model.
      - Draw bounding boxes and labels on each frame.
      - Overlay per-frame vehicle count and density level.
      - Save the annotated video(s) under output_dir.

    Returns a results dict with:
```

```
        'vehicle_counts'      - list[int]
        'density_levels'      - list[str]
        'classification_log'  - per-frame list of class names
        'output_files'        - dict of video paths
        'processed_frames'    - number of frames processed
        'fps'                 - frames per second of input video
    """
    input_path = Path(input_path)
    if not input_path.exists():
        raise FileNotFoundError(f"Input video not found: {input_path}")

    output_dir = Path(output_dir)
    output_dir.mkdir(parents=True, exist_ok=True)

    cap = cv2.VideoCapture(str(input_path))
    if not cap.isOpened():
        raise RuntimeError(f"Could not open video: {input_path}")

    fps = cap.get(cv2.CAP_PROP_FPS) or 25.0

    # Read first frame to get reliable width/height
    ret, frame_bgr = cap.read()
    if not ret:
        cap.release()
        raise RuntimeError("Could not read first frame from video.")

    height, width = frame_bgr.shape[:2]
    print(f"Processing video: {input_path.name}")
    print(f"Resolution: {width}x{height}, FPS: {fps:.1f}")

    fourcc = cv2.VideoWriter_fourcc(*"mp4v")

    analysis_path = output_dir / f"{input_path.stem}_analysis.mp4"
    detection_only_path = output_dir / f"{input_path.stem}_detections.mp4"
    mask_path = output_dir / f"{input_path.stem}_mask.mp4"

    writer_analysis = cv2.VideoWriter(str(analysis_path), fourcc, fps, (width,␣
↪height))
    writer_detections = cv2.VideoWriter(str(detection_only_path), fourcc, fps,␣
↪(width, height))
    writer_mask = cv2.VideoWriter(str(mask_path), fourcc, fps, (width, height))

    bg_subtractor = create_background_subtractor()

    vehicle_counts = []
    density_levels = []
    classification_log = []
```

```
frame_idx = 0
start_time = time.time()

while ret:
    boxes, fg_mask, mask_clean = detect_vehicles_in_frame(
        frame_bgr, bg_subtractor, min_area=min_area
    )

    annotated = frame_bgr.copy()
    detections_only = frame_bgr.copy()
    mask_color = cv2.cvtColor(mask_clean, cv2.COLOR_GRAY2BGR)

    frame_classes = []
    count = 0

    for (x1, y1, x2, y2) in boxes:
        crop_bgr = frame_bgr[y1:y2, x1:x2]
        pred_class, conf, pred_idx, probs = classify_vehicle_crop(
            model, crop_bgr, device=device
        )
        if pred_class is None:
            continue

        count += 1
        frame_classes.append(pred_class)

        # Draw bounding boxes
        cv2.rectangle(annotated, (x1, y1), (x2, y2), (0, 255, 0), 2)
        cv2.rectangle(detections_only, (x1, y1), (x2, y2), (0, 255, 0), 2)

        # Label string
        label = f"{pred_class} {conf:.2f}"
        label_y = max(y1 - 5, 0)
        cv2.putText(
            annotated,
            label,
            (x1, label_y),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5,
            (0, 0, 0),
            1,
            cv2.LINE_AA,
        )

    density = compute_density_level(count, thresholds=density_thresholds)
```

```
        # Overlay per-frame summary text
        overlay_text = f"Frame: {frame_idx} | Vehicles: {count} | Density:␣
↪{density}"
        cv2.putText(
            annotated,
            overlay_text,
            (10, 25),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.7,
            (0, 255, 255),
            2,
            cv2.LINE_AA,
        )

        vehicle_counts.append(count)
        density_levels.append(density)
        classification_log.append(frame_classes)

        writer_analysis.write(annotated)
        writer_detections.write(detections_only)
        writer_mask.write(mask_color)

        frame_idx += 1
        ret, frame_bgr = cap.read()  # read next frame

    cap.release()
    writer_analysis.release()
    writer_detections.release()
    writer_mask.release()

    elapsed = time.time() - start_time
    print(f"Done. Processed {frame_idx} frames in {elapsed:.1f} seconds.")

    return {
        "vehicle_counts": vehicle_counts,
        "density_levels": density_levels,
        "classification_log": classification_log,
        "output_files": {
            "analysis": analysis_path,
            "detections": detection_only_path,
            "mask": mask_path,
        },
        "processed_frames": frame_idx,
        "fps": fps,
    }
```

Run the pipeline on a real video run the pipeline on a real traffic video:

- **Input** video: `data/real_life/traffic_example.mov`
- **Output** videos: written under `outputs/videos/`

[7]:
```python
# Default path for real-world video
# (Place your file here: data/real_life/traffic_example.mov)
INPUT_VIDEO_PATH = REAL_LIFE_DATA / "traffic_example.mov"

print("Input video path:", INPUT_VIDEO_PATH)

if not INPUT_VIDEO_PATH.exists():
    print(
        "WARNING: Input video file not found.\n"
        "Please place a file named 'traffic_example.mov' inside:\n"
        f"    {REAL_LIFE_DATA}\n\n"
        "Or adjust INPUT_VIDEO_PATH in this cell."
    )
    results = None
else:
    results = process_traffic_video(
        input_path=INPUT_VIDEO_PATH,
        output_dir=VIDEOS_DIR,
        model=model,
        idx_to_class=idx_to_class,
        device=DEVICE,
        target_size=TARGET_SIZE,
        min_area=800,
        density_thresholds=(3, 7),
    )

    print("\nOutput videos written to:")
    for name, path in results["output_files"].items():
        print(f"  {name}: {path}")
```

```
Input video path: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-project-g3/data/real_life/traffic_example.mov
Processing video: traffic_example.mov
Resolution: 640x372, FPS: 60.0
Done. Processed 1977 frames in 9.7 seconds.

Output videos written to:
  analysis: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-
project-g3/outputs/videos/traffic_example_analysis.mp4
  detections: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-
project-g3/outputs/videos/traffic_example_detections.mp4
  mask: /Users/matthashemi/Documents/Personal/University/MS-AAI-
Courses/07-AAI-521/aai-521-final-
```

project-g3/outputs/videos/traffic_example_mask.mp4

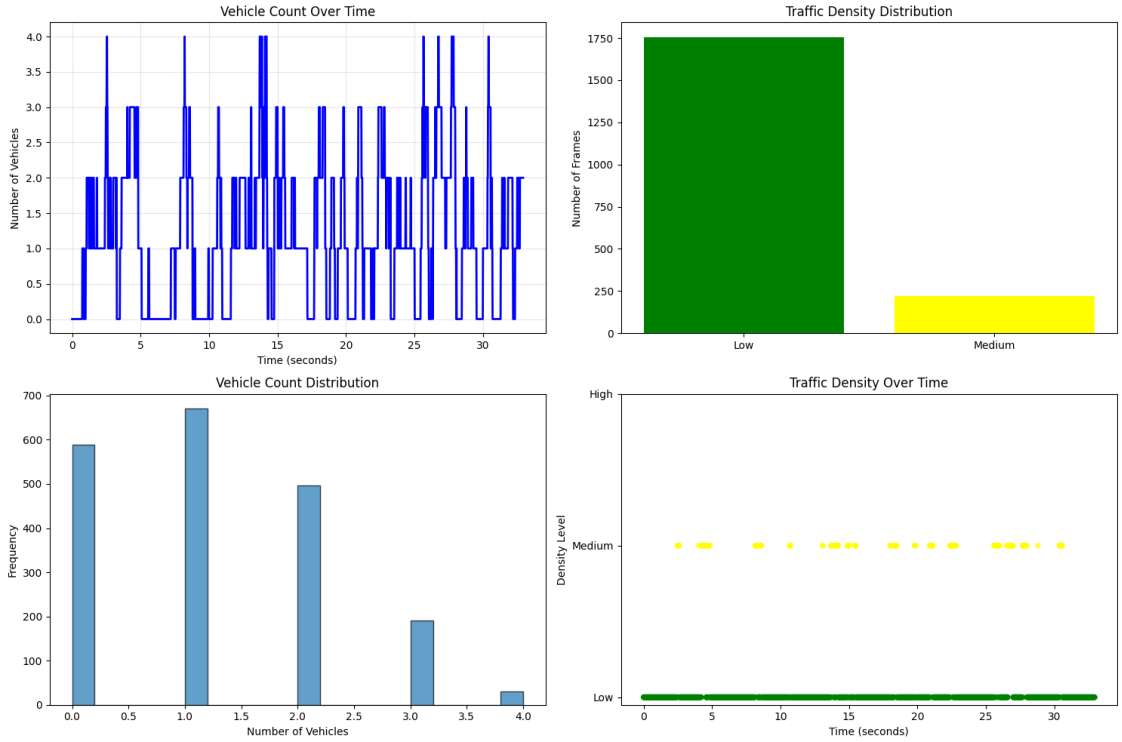
Traffic Analytics and Visualizations

- Plot the estimated number of vehicles over time,
- Visualize traffic density distributions and evolution,
- Summarize basic statistics (e.g., average vehicles per frame).

[8]:
```python
# Visualize results and analytics
if results is not None:
    vehicle_counts = results["vehicle_counts"]
    density_levels = results["density_levels"]

    if len(vehicle_counts) == 0:
        print("No vehicle counts recorded - nothing to visualize.")
    else:
        # Create analytics plots
        fig, axes = plt.subplots(2, 2, figsize=(15, 10))

        # Time axis in seconds using actual FPS
        fps = results.get("fps", 10.0)
        time_axis = np.arange(len(vehicle_counts)) / fps

        # Plot 1: Vehicle count over time
        axes[0, 0].plot(time_axis, vehicle_counts, "b-", linewidth=2)
        axes[0, 0].set_title("Vehicle Count Over Time")
        axes[0, 0].set_xlabel("Time (seconds)")
        axes[0, 0].set_ylabel("Number of Vehicles")
        axes[0, 0].grid(True, alpha=0.3)

        # Plot 2: Density distribution
        density_counts = {}
        for level in density_levels:
            density_counts[level] = density_counts.get(level, 0) + 1

        # Keep color mapping consistent for Low/Medium/High
        bar_labels = list(density_counts.keys())
        bar_colors = []
        for level in bar_labels:
            if level.lower() == "low":
                bar_colors.append("green")
            elif level.lower() == "medium":
                bar_colors.append("yellow")
            else:  # treat anything else as "High"
                bar_colors.append("red")

        axes[0, 1].bar(bar_labels, density_counts.values(), color=bar_colors)
        axes[0, 1].set_title("Traffic Density Distribution")
```

```
        axes[0, 1].set_ylabel("Number of Frames")

        # Plot 3: Vehicle count histogram
        axes[1, 0].hist(vehicle_counts, bins=20, alpha=0.7, edgecolor="black")
        axes[1, 0].set_title("Vehicle Count Distribution")
        axes[1, 0].set_xlabel("Number of Vehicles")
        axes[1, 0].set_ylabel("Frequency")

        # Plot 4: Density over time as a scatter
        density_numeric = []
        for level in density_levels:
            if level.lower() == "low":
                density_numeric.append(0)
            elif level.lower() in ("medium", "average"):
                density_numeric.append(1)
            else:  # high
                density_numeric.append(2)

        colors = ["green" if d == 0 else "yellow" if d == 1 else "red" for d in␣
↪density_numeric]
        axes[1, 1].scatter(time_axis, density_numeric, c=colors, alpha=0.7,␣
↪s=20)
        axes[1, 1].set_title("Traffic Density Over Time")
        axes[1, 1].set_xlabel("Time (seconds)")
        axes[1, 1].set_ylabel("Density Level")
        axes[1, 1].set_yticks([0, 1, 2])
        axes[1, 1].set_yticklabels(["Low", "Medium", "High"])

        plt.tight_layout()
        plt.show()

        # Print summary statistics
        print("\n=== TRAFFIC ANALYSIS SUMMARY ===")
        total_frames = len(vehicle_counts)
        duration = total_frames / fps if fps > 0 else float("nan")
        print(f"Total frames processed: {total_frames}")
        print(f"Video duration: {duration:.1f} seconds")
        print(f"Average vehicles per frame: {np.mean(vehicle_counts):.1f}")
        print(f"Maximum vehicles detected: {np.max(vehicle_counts)}")
        print(f"Minimum vehicles detected: {np.min(vehicle_counts)}")

        print("\nDensity Distribution:")
        for level, count in density_counts.items():
            percentage = (count / total_frames) * 100
            print(f"  {level}: {count} frames ({percentage:.1f}%)")

        print("\nOutput Files Generated:")
```

```
        for i, (name, file_path) in enumerate(results["output_files"].items(),␣
↪1):
            file_path = Path(file_path)
            if file_path.exists():
                file_size = file_path.stat().st_size / (1024 * 1024)  # MB
                print(f"  {i}. {file_path.name} ({file_size:.1f} MB)")
            else:
                print(f"  {i}. {file_path.name} (file not found)")
else:
    print("No results to visualize. Please run the video processing cell first.
↪")
```

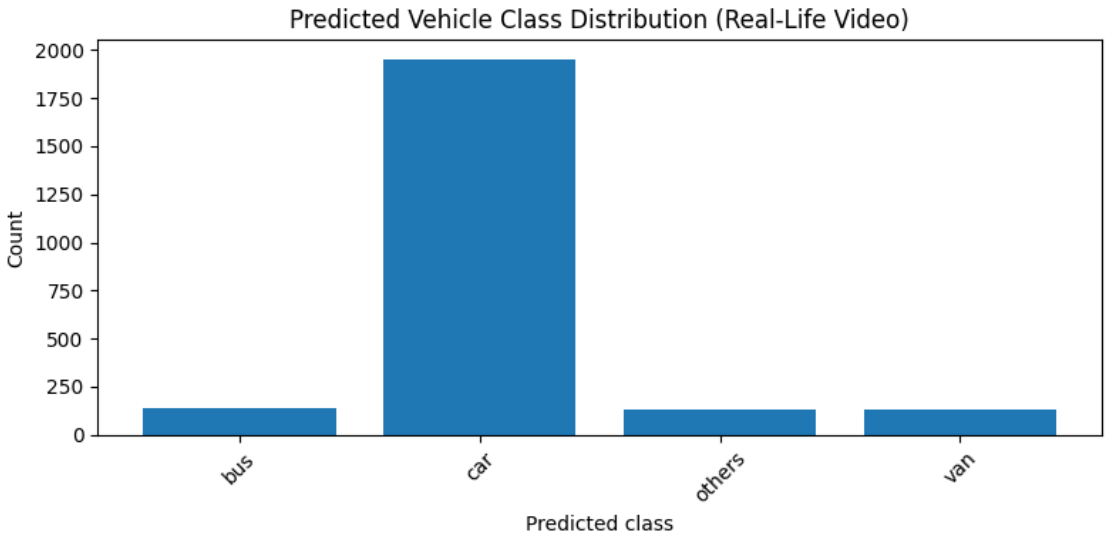



```
=== TRAFFIC ANALYSIS SUMMARY ===
Total frames processed: 1977
Video duration: 33.0 seconds
Average vehicles per frame: 1.2
Maximum vehicles detected: 4
Minimum vehicles detected: 0

Density Distribution:
  Low: 1755 frames (88.8%)
  Medium: 222 frames (11.2%)
```

```
Output Files Generated:
  1. traffic_example_analysis.mp4 (11.9 MB)
  2. traffic_example_detections.mp4 (10.5 MB)
  3. traffic_example_mask.mp4 (9.5 MB)
```

Predicted class distribution in the real-life video

[9]:
```python
if results is not None:
    all_predicted_classes = [
        cls
        for frame_classes in results["classification_log"]
        for cls in frame_classes
    ]

    if len(all_predicted_classes) == 0:
        print("No predicted vehicles to analyze for class distribution.")
    else:
        unique_classes, counts = np.unique(all_predicted_classes,␣
 ↪return_counts=True)
        class_counts = dict(zip(unique_classes, counts))

        plt.figure(figsize=(8, 4))
        plt.bar(class_counts.keys(), class_counts.values())
        plt.xlabel("Predicted class")
        plt.ylabel("Count")
        plt.title("Predicted Vehicle Class Distribution (Real-Life Video)")
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

        print("Predicted class counts:")
        for cls in sorted(class_counts.keys()):
            print(f"  {cls}: {class_counts[cls]}")
else:
    print("No results to analyze. Please run the video processing cell first.")
```




```
Predicted class counts:
  bus: 140
  car: 1955
  others: 133
  van: 133
```

Preview the generated videos inline

[10]:
```
if results is not None:
    from IPython.display import Video, display, HTML

    for name, path in results["output_files"].items():
        if Path(path).exists():
            display(HTML(f"<h3>{name.title()} Video</h3>"))
            display(Video(str(path), width=640, height=480))
        else:
            print(f"Video file not found on disk: {path}")
else:
    print("No videos to display. Please run the processing cell first.")
```

```
<IPython.core.display.HTML object>

<IPython.core.display.Video object>

<IPython.core.display.HTML object>

<IPython.core.display.Video object>

<IPython.core.display.HTML object>

<IPython.core.display.Video object>
```

1.1 Summary and Conclusions

In this notebook, we demonstrated how to **deploy our trained vehicle classifier on a real-life traffic video**:

- We reused the `VehicleClassifier` model trained on the **DETRAC** dataset and the same class mapping from our cropped vehicle dataset.
- Using **background subtraction**, we automatically detected moving vehicles in each frame of a real-world video.
- For each detected region, we:
  - Cropped and resized the patch,
  - Classified it into one of our vehicle classes,
  - Drew bounding boxes and labels with confidence scores.
- We overlaid a simple **traffic density estimate** per frame (Low / Medium / High).
- The resulting annotated videos were saved under:
  `outputs/videos/`
- We generated **time-series and distribution plots**, summarizing how vehicle counts and density levels evolved over the duration of the video.

This notebook serves as a **bridge between the experimental pipeline (Notebooks 01–05)** and a **practical real-world application**:

- It confirms that our model can be used on real videos, not just on curated DETRAC sequences.
- It also highlights limitations:
  - Background subtraction is sensitive to camera motion and lighting changes.
  - Detection quality directly impacts classification accuracy.

Possible next steps:

- Replace background subtraction with a more robust detector (e.g., YOLO-style model).
- Improve tracking across frames to obtain stable vehicle IDs and trajectories.
- Integrate this pipeline more tightly with the counting and LMV/HMV analytics from `04_vehicle_counting_and_analysis.ipynb`.

For the purposes of this project, this notebook is our **deployment-style demo** and completes the end-to-end story:
from **dataset exploration → training and evaluation → real-world inference and analytics**.

Vehicle Classification Model (Tensorflow) - 20% dataset

What This Model Does

This CNN (Convolutional Neural Network) model classifies vehicles into 4 categories:

- **Car** - Regular passenger vehicles
- **Van** - Minivans and cargo vans
- **Bus** - Large buses and coaches
- **Others** - other vehicles

The model then groups these into 2 main types:

- **LMV (Light Motor Vehicles)** - Cars, vans, and others
- **HMV (Heavy Motor Vehicles)** - Buses and trucks

Dataset Used

We use the **UA-DETRAC dataset** which contains traffic videos from highways and intersections.

Why Only 20% of the Dataset?

The full dataset is HUGE -

- **20% of sequences** = 12 out of 60 video sequences
- **Total samples created**: 96,570 vehicle images
- **Processing time**: Faster processing time

> **Note:** This notebook is an **experimental TensorFlow + YOLO lab**. The final project pipeline uses the **PyTorch** notebooks: `03_vehicle_classification_model.ipynb`, `04_vehicle_counting_and_analysis.ipynb`, and `06_real_life_traffic_video_demo.ipynb`.

In [ ]:
```
# Vehicle Classification — TensorFlow Optimized (20% Dataset)

import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import xml.etree.ElementTree as ET
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
```

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Dataset paths
DETRAC_ROOT = "./UA-DETRAC"
IMAGES_PATH = Path(DETRAC_ROOT) / "images"
TRAIN_ANNOT_PATH = Path(DETRAC_ROOT) / "annotations" / "DETRAC-Train-Annotat
TEST_ANNOT_PATH = Path(DETRAC_ROOT) / "annotations" / "DETRAC-Test-Annotatic
TARGET_SIZE = (64, 64)

print(f"DETRAC_ROOT: {DETRAC_ROOT}")
print(f"Images path: {IMAGES_PATH}")
print(f"Train annotations: {TRAIN_ANNOT_PATH}")
print(f"Test annotations: {TEST_ANNOT_PATH}")

# Check GPU availability
print(f"\n{'='*60}")
print("GPU CONFIGURATION")
print(f"{'='*60}")
print(f"TensorFlow version: {tf.__version__}")
print(f"GPU Available: {tf.config.list_physical_devices('GPU')}")
if tf.config.list_physical_devices('GPU'):
    print("✓ Training will use GPU")
else:
    print("⚠ Training will use CPU (slower)")
```

```
DETRAC_ROOT: ./UA-DETRAC
Images path: UA-DETRAC/images
Train annotations: UA-DETRAC/annotations/DETRAC-Train-Annotations-XML/DETRAC
-Train-Annotations-XML
Test annotations: UA-DETRAC/annotations/DETRAC-Test-Annotations-XML/DETRAC-T
est-Annotations-XML

============================================================
GPU CONFIGURATION
============================================================
TensorFlow version: 2.20.0
GPU Available: [PhysicalDevice(name='/physical_device:GPU:0', device_type='G
PU')]
✓ Training will use GPU
```

Generate Training Dataset from DETRAC

What This Does

This function extracts individual vehicle images from the DETRAC video sequences. It reads the XML annotation files to find where each vehicle is located in each frame, then crops out that vehicle and saves it as a training sample.

Why Only 15% of Sequences?

- **Full dataset is too large**: Loading all sequences would need take alot loading time
- **Solution**: Use only 15% (9 out of 60 sequences) to stay within memory limits
- **Result**: Still get 96,570 vehicle samples - plenty for training!

How It Works

1. **Find XML files**: Lists all annotation files in the DETRAC folder
2. **Select 15%**: Randomly picks 9 sequences to process
3. **For each sequence**:
   - Reads the XML file to find vehicle locations
   - Opens each frame (image) from that sequence
   - Crops out each vehicle using the bounding box coordinates
   - Resizes the crop to 64×64 pixels (our model's input size)
4. **Normalize images**: Converts pixel values from 0-255 to 0-1 range
5. **Create labels**: Assigns each crop to its vehicle type (car, van, bus, others)

What We Get

- **Images**: Numpy array of 64×64×3 vehicle crops
- **Labels**: Array of class indices (0=car, 1=van, 2=others, 3=bus)
- **Class mapping**: Dictionary linking class names to numbers

This dataset is ready to feed directly into our CNN for training!

In [ ]:
```
# Dataset Generation (20% of sequences)

def generate_cnn_training_dataset(annotations_dir, images_dir, target_size=T

    # Convert to Path objects for easier file handling
    annotations_dir = Path(annotations_dir)
    images_dir = Path(images_dir)

    # Check if folders exist
    if not annotations_dir.exists():
        raise FileNotFoundError(f"Annotations directory not found: {annotati
    if not images_dir.exists():
        raise FileNotFoundError(f"Images directory not found: {images_dir}")

    print(f"\n{'='*60}")
    print(f"GENERATING CNN TRAINING DATASET")
    print(f"{'='*60}")
    print(f"Using {use_percentage*100:.0f}% of sequences for training")

    # Storage for our dataset
    crops, label_indices, metadata = [], [], []
    class_to_idx = {}

    # Get all XML files and select percentage
    xml_files = sorted(annotations_dir.glob("*.xml"))
    num_sequences = int(len(xml_files) * use_percentage)
```

```
    xml_files = xml_files[:num_sequences]

    print(f"Total sequences available: {len(sorted(annotations_dir.glob('*.x
    print(f"Using {num_sequences} sequences ({use_percentage*100:.0f}%)")
    print("Processing sequences...")

    # Process each sequence (video)
    for idx, xml_path in enumerate(xml_files, 1):
        seq_name = xml_path.stem
        seq_image_dir = images_dir / seq_name

        # Skip if image folder doesn't exist
        if not seq_image_dir.exists():
            continue

        # Print progress every 5 sequences
        if idx % 5 == 0 or idx == 1:
            print(f"  [{idx}/{num_sequences}] Processing: {seq_name}")

        # Parse the XML annotation file
        tree = ET.parse(str(xml_path))

        # Go through each frame in the sequence
        for frame in tree.findall(".//frame"):
            frame_num = int(frame.attrib["num"])
            img_path = seq_image_dir / f"img{frame_num:05d}.jpg"

            if not img_path.exists():
                continue

            image = cv2.imread(str(img_path))
            if image is None:
                continue
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            h_img, w_img = image.shape[:2]

            target_list = frame.find("target_list")
            if target_list is None:
                continue

            for target in target_list.findall("target"):
                attr = target.find("attribute")
                vehicle_type = attr.attrib.get("vehicle_type", "unknown").lo

                # Initialize class mapping
                if vehicle_type not in class_to_idx:
                    class_to_idx[vehicle_type] = len(class_to_idx)

                box = target.find("box")
                if box is None:
                    continue

                # Extract bounding box
                left = max(int(float(box.attrib["left"])), 0)
                top = max(int(float(box.attrib["top"])), 0)
                width = int(float(box.attrib["width"]))
```

```
                height = int(float(box.attrib["height"]))
                x2 = min(left + width, w_img)
                y2 = min(top + height, h_img)

                if x2 <= left or y2 <= top:
                    continue

                # Crop and resize vehicle
                crop = image[top:y2, left:x2]
                if crop.size == 0:
                    continue
                resized = cv2.resize(crop, target_size, interpolation=cv2.IN

                # Store crop and label
                label_idx = class_to_idx[vehicle_type]
                crops.append(resized)
                label_indices.append(label_idx)
                metadata.append({
                    "sequence": seq_name,
                    "frame": frame_num,
                    "vehicle_type": vehicle_type,
                    "label_idx": label_idx,
                })

    # Make sure we got some samples
    if not crops:
        raise RuntimeError("No samples were extracted; check paths and annot

    # Convert to numpy arrays
    cnn_images = np.stack(crops).astype("float32") / 255.0  # Normalize to [
    cnn_labels = np.array(label_indices, dtype="int32")

    # Create class name mapping
    idx_to_class = {idx: name for name, idx in class_to_idx.items()}


    # Print summary
    print(f"\n{'='*60}")
    print("DATASET SUMMARY")
    print(f"{'='*60}")
    print(f"Total samples: {len(cnn_labels):,}")
    print(f"Image shape: {cnn_images.shape}")
    print(f"Number of classes: {len(class_to_idx)}")
    print(f"\nClass distribution:")
    for class_name, class_idx in class_to_idx.items():
        count = np.sum(cnn_labels == class_idx)
        percentage = (count / len(cnn_labels)) * 100
        print(f"  {class_name:12s} (class {class_idx}): {count:7,} samples (

    return {
        "images": cnn_images,
        "labels": cnn_labels,
        "class_to_idx": class_to_idx,
        "idx_to_class": idx_to_class,
        "metadata": metadata,
```

```
        "num_classes": len(class_to_idx)
    }
```

In [ ]:
```
# Generate Dataset
print("\nStarting dataset generation...")
cnn_dataset = generate_cnn_training_dataset(TRAIN_ANNOT_PATH, IMAGES_PATH, u

X = cnn_dataset["images"]
y = cnn_dataset["labels"]
class_to_idx = cnn_dataset["class_to_idx"]
idx_to_class = cnn_dataset["idx_to_class"]
num_classes = cnn_dataset["num_classes"]
```

```
Starting dataset generation...

============================================================
GENERATING CNN TRAINING DATASET
============================================================
Using 20% of sequences for training
Total sequences available: 60
Using 12 sequences (20%)
Processing sequences...
  [1/12] Processing: MVI_20011
  [5/12] Processing: MVI_20034
  [10/12] Processing: MVI_20062

============================================================
DATASET SUMMARY
============================================================
Total samples: 96,570
Image shape: (96570, 64, 64, 3)
Number of classes: 4

Class distribution:
  car          (class 0):  81,232 samples (84.12%)
  van          (class 1):   7,702 samples ( 7.98%)
  others       (class 2):     518 samples ( 0.54%)
  bus          (class 3):   7,118 samples ( 7.37%)
```

Balance Dataset

The Problem We're Solving

Our dataset has a **severe class imbalance**:

- Cars: (84%) - Too many!
- Van: (8%)
- Bus: (8%)
- Others: (0.5%) - Almost nothing!

If we train on this imbalanced data, the model will just predict "car" for everything and ignore the other types.

Our Solution

We use a two-part approach:

Part 1: Balance the Big Three Classes

- Take the smallest of car/van/bus (which is bus with 7,118 samples)
- Randomly select 7,118 samples from car (out of 81,232)
- Randomly select 7,118 samples from van (out of 7,702)
- Keep all 7,118 bus samples
- This gives us balanced data without throwing away too much

Part 2: Keep ALL "Others" + Use Weights

- We can't afford to lose any "others" samples (only have 518!)
- Keep all 518 "others" samples
- Give "others" a ** higher weight** during training
- This tells the model: "Getting 'others' wrong is much worse than getting car/van/bus wrong"

In [ ]:
```
# Balance Classes

print(f"\n{'='*60}")
print("BALANCING CLASSES (HYBRID APPROACH)")
print(f"{'='*60}")

# Count samples per class
class_counts = {i: np.sum(y == i) for i in range(num_classes)}
print("\nOriginal class distribution:")
for class_idx, count in class_counts.items():
    print(f"  {idx_to_class[class_idx]:12s}: {count:7,} samples")

# Identify the "others" class (smallest class)
minority_class_idx = min(class_counts, key=class_counts.get)
minority_class_name = idx_to_class[minority_class_idx]

print(f"\nMinority class: '{minority_class_name}' with {class_counts[minorit

# Find target size (smallest class EXCLUDING "others")
target_sizes = {idx: count for idx, count in class_counts.items() if idx !=
target_class_size = min(target_sizes.values())
target_class_name = idx_to_class[min(target_sizes, key=target_sizes.get)]

print(f"Target balance size: {target_class_size:,} samples (from '{target_cl
print(f"\nStrategy:")
print(f"  1. Balance car, van, bus to {target_class_size:,} samples each")
print(f"  2. Keep all {class_counts[minority_class_idx]:,} '{minority_class_
print(f"  3. Use class weights to compensate for '{minority_class_name}' imb

# Undersample each class (except minority) to match target
balanced_indices = []
```

```python
for class_idx in range(num_classes):
    class_indices = np.where(y == class_idx)[0]

    if class_idx == minority_class_idx:
        # Keep ALL samples from minority class
        sampled_indices = class_indices
        print(f"  ✓ Keeping all {len(class_indices):,} '{idx_to_class[class_
    else:
        # Undersample to target size
        np.random.seed(42)
        sampled_indices = np.random.choice(class_indices, size=target_class_
        print(f"  ✓ Sampling {target_class_size:,} from {len(class_indices):

    balanced_indices.extend(sampled_indices)

# Shuffle balanced indices
np.random.seed(42)
np.random.shuffle(balanced_indices)

# Create balanced dataset
X_balanced = X[balanced_indices]
y_balanced = y[balanced_indices]

print(f"\nBalanced dataset:")
print(f"  Total samples: {len(y_balanced):,} (was {len(y):,})")
print(f"  Data reduction: {(1 - len(y_balanced)/len(y))*100:.1f}%")

# Verify final distribution
print(f"\nFinal class distribution:")
final_counts = {}
for class_idx in range(num_classes):
    count = np.sum(y_balanced == class_idx)
    final_counts[class_idx] = count
    print(f"  {idx_to_class[class_idx]:12s}: {count:7,} samples")

# ============================================
# Calculate Class Weights for Training
# ============================================
print(f"\n{'='*60}")
print("CALCULATING CLASS WEIGHTS")
print(f"{'='*60}")

# Calculate weights to compensate for class imbalance
total_samples = len(y_balanced)
class_weights = {}

for class_idx in range(num_classes):
    class_count = final_counts[class_idx]
    weight = total_samples / (num_classes * class_count)
    class_weights[class_idx] = weight

print("\nClass weights (to compensate for imbalance):")
for class_idx, weight in class_weights.items():
    print(f"  {idx_to_class[class_idx]:12s}: {weight:.4f}")

print(f"\nNote: Higher weight for '{minority_class_name}' = model focuses mo
```

```
============================================================
BALANCING CLASSES (HYBRID APPROACH)
============================================================

Original class distribution:
  car         :  81,232 samples
  van         :   7,702 samples
  others      :     518 samples
  bus         :   7,118 samples

Minority class: 'others' with 518 samples
Target balance size: 7,118 samples (from 'bus' class)

Strategy:
  1. Balance car, van, bus to 7,118 samples each
  2. Keep all 518 'others' samples
  3. Use class weights to compensate for 'others' imbalance
  ✓ Sampling 7,118 from 81,232 'car' samples
  ✓ Sampling 7,118 from 7,702 'van' samples
  ✓ Keeping all 518 'others' samples
  ✓ Sampling 7,118 from 7,118 'bus' samples

Balanced dataset:
  Total samples: 21,872 (was 96,570)
  Data reduction: 77.4%

Final class distribution:
  car         :   7,118 samples
  van         :   7,118 samples
  others      :     518 samples
  bus         :   7,118 samples

============================================================
CALCULATING CLASS WEIGHTS
============================================================

Class weights (to compensate for imbalance):
  car         : 0.7682
  van         : 0.7682
  others      : 10.5560
  bus         : 0.7682

Note: Higher weight for 'others' = model focuses more on this class during t
raining
```

In [ ]:

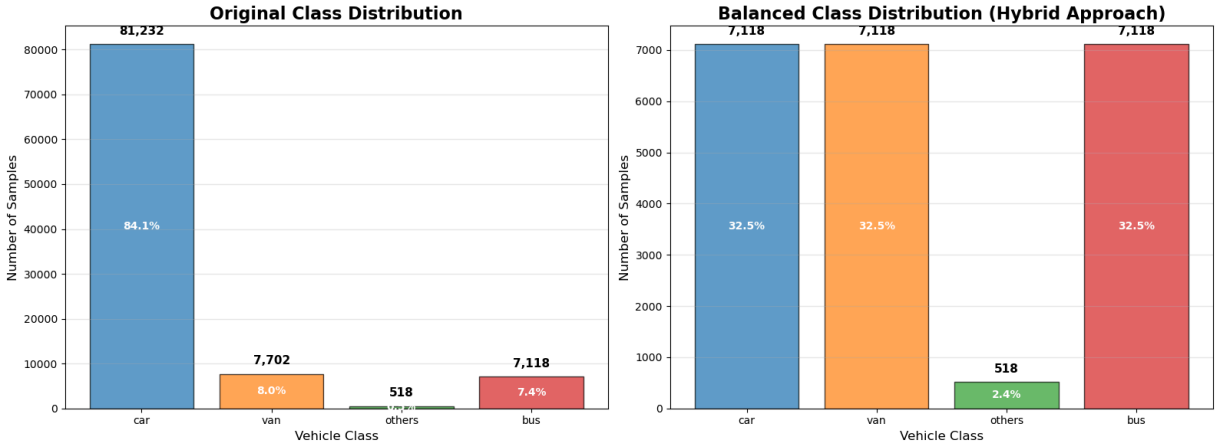
```python
# Visualize Class Distribution and Sample Images
import matplotlib.pyplot as plt
import numpy as np

print(f"\n{'='*60}")
print("VISUALIZING CLASS DISTRIBUTION AND SAMPLES")
print(f"{'='*60}")

# Plot Class Distribution (Before and After Balancing)

fig, axes = plt.subplots(1, 2, figsize=(16, 6))
```

```
# Original distribution
class_names_list = [idx_to_class[i] for i in range(num_classes)]
original_counts = [class_counts[i] for i in range(num_classes)]
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

axes[0].bar(class_names_list, original_counts, color=colors, alpha=0.7, edge
axes[0].set_title('Original Class Distribution', fontsize=16, fontweight='bo
axes[0].set_xlabel('Vehicle Class', fontsize=12)
axes[0].set_ylabel('Number of Samples', fontsize=12)
axes[0].grid(axis='y', alpha=0.3)

# Add value labels on bars
for i, (name, count) in enumerate(zip(class_names_list, original_counts)):
    axes[0].text(i, count + max(original_counts)*0.02, f'{count:,}',
                 ha='center', va='bottom', fontweight='bold', fontsize=11)
    percentage = (count / sum(original_counts)) * 100
    axes[0].text(i, count/2, f'{percentage:.1f}%',
                 ha='center', va='center', fontsize=10, color='white', fontwe

# Balanced distribution
balanced_counts = [final_counts[i] for i in range(num_classes)]

axes[1].bar(class_names_list, balanced_counts, color=colors, alpha=0.7, edge
axes[1].set_title('Balanced Class Distribution (Hybrid Approach)', fontsize=
axes[1].set_xlabel('Vehicle Class', fontsize=12)
axes[1].set_ylabel('Number of Samples', fontsize=12)
axes[1].grid(axis='y', alpha=0.3)

# Add value labels on bars
for i, (name, count) in enumerate(zip(class_names_list, balanced_counts)):
    axes[1].text(i, count + max(balanced_counts)*0.02, f'{count:,}',
                 ha='center', va='bottom', fontweight='bold', fontsize=11)
    percentage = (count / sum(balanced_counts)) * 100
    axes[1].text(i, count/2, f'{percentage:.1f}%',
                 ha='center', va='center', fontsize=10, color='white', fontwe

plt.tight_layout()
plt.show()

# Show 5 Sample Images from Each Class

print("\nGenerating sample images for each class...")

# Number of samples to show per class
n_samples = 5

# Create figure with subplots
fig, axes = plt.subplots(num_classes, n_samples, figsize=(15, 3*num_classes)

# Handle case where there's only one class
if num_classes == 1:
    axes = axes.reshape(1, -1)

for class_idx in range(num_classes):
    # Get indices for this class from balanced dataset
```

```
    class_mask = y_balanced == class_idx
    class_samples_indices = np.where(class_mask)[0]

    # Randomly select n_samples (or fewer if not enough samples)
    n_available = min(n_samples, len(class_samples_indices))
    np.random.seed(42)
    selected_indices = np.random.choice(class_samples_indices, size=n_availa

    print(f"  {idx_to_class[class_idx]:12s}: Showing {n_available} samples")

    for col_idx in range(n_samples):
        ax = axes[class_idx, col_idx]

        if col_idx < n_available:
            # Get image
            img = X_balanced[selected_indices[col_idx]]

            # Display image
            ax.imshow(img)
            ax.axis('off')

            # Add title to first image in row
            if col_idx == 0:
                ax.set_title(f'{idx_to_class[class_idx].upper()}',
                             fontsize=14, fontweight='bold', loc='left', pad=1
        else:
            # No image available
            ax.axis('off')
            if col_idx == 0:
                ax.text(0.5, 0.5, 'Not enough\nsamples',
                        ha='center', va='center', fontsize=10)

# Overall title
fig.suptitle('Sample Images from Each Class (After Balancing)',
             fontsize=16, fontweight='bold', y=0.995)

plt.tight_layout()
plt.show()

# Summary Statistics Table

print(f"\n{'='*60}")
print("CLASS DISTRIBUTION SUMMARY")
print(f"{'='*60}")

print(f"\n{'Class':<12} {'Original':<12} {'Balanced':<12} {'Reduction':<12}
print("-" * 60)

for class_idx in range(num_classes):
    class_name = idx_to_class[class_idx]
    orig_count = class_counts[class_idx]
    balanced_count = final_counts[class_idx]
    reduction = ((orig_count - balanced_count) / orig_count) * 100
    weight = class_weights[class_idx]

    print(f"{class_name:<12} {orig_count:<12,} {balanced_count:<12,} {reduct
```

```
print("-" * 60)
print(f"{'TOTAL':<12} {sum(class_counts.values()):<12,} {sum(final_counts.va
      f"{((sum(class_counts.values()) - sum(final_counts.values())) / sum(cl

print(f"\n{'='*60}")


# Class Weight Visualization

print("\nGenerating class weight visualization...")

fig, ax = plt.subplots(1, 1, figsize=(10, 6))

weights_list = [class_weights[i] for i in range(num_classes)]
colors_weight = ['#2ca02c' if w < 2 else '#ff7f0e' if w < 10 else '#d62728'

bars = ax.bar(class_names_list, weights_list, color=colors_weight, alpha=0.7

ax.set_title('Class Weights (Higher = More Focus During Training)',
             fontsize=16, fontweight='bold')
ax.set_xlabel('Vehicle Class', fontsize=12)
ax.set_ylabel('Weight', fontsize=12)
ax.grid(axis='y', alpha=0.3)

# Add value labels on bars
for i, (name, weight) in enumerate(zip(class_names_list, weights_list)):
    ax.text(i, weight + max(weights_list)*0.02, f'{weight:.2f}',
            ha='center', va='bottom', fontweight='bold', fontsize=11)

# Add horizontal line at weight=1 (balanced)
ax.axhline(y=1.0, color='gray', linestyle='--', linewidth=2, alpha=0.5, labe
ax.legend()

plt.tight_layout()
plt.show()
```

```
============================================================
VISUALIZING CLASS DISTRIBUTION AND SAMPLES
============================================================
✓ Saved class distribution plot: class_distribution.png
```



```
Generating sample images for each class...
  car         : Showing 5 samples
  van         : Showing 5 samples
  others      : Showing 5 samples
  bus         : Showing 5 samples
✓ Saved sample images: class_samples.png
```

Sample Images from Each Class (After Balancing)

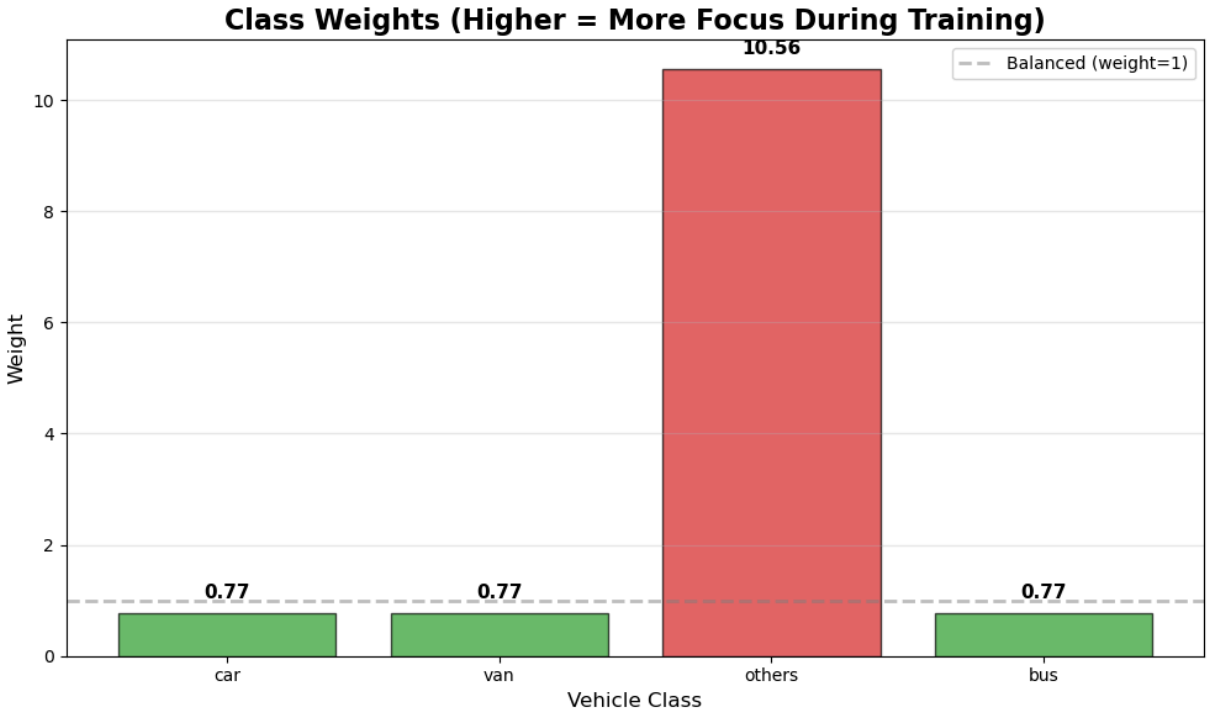
CAR

VAN

OTHERS

BUS

```
============================================================
CLASS DISTRIBUTION SUMMARY
============================================================

Class        Original     Balanced     Reduction    Weight
------------------------------------------------------------
car          81,232       7,118             91.2%    0.7682
van          7,702        7,118              7.6%    0.7682
others       518          518                0.0%   10.5560
bus          7,118        7,118              0.0%    0.7682
------------------------------------------------------------
TOTAL        96,570       21,872            77.4%

============================================================

Generating class weight visualization...
✓ Saved class weights plot: class_weights.png
```




```
============================================================
✓ ALL VISUALIZATIONS COMPLETE!
============================================================

Generated files:
  1. class_distribution.png - Before/after balancing comparison
  2. class_samples.png - 5 sample images per class
  3. class_weights.png - Training weights visualization
```

In [ ]:
```python
# Train/Validation Split

print(f"\n{'='*60}")
print("SPLITTING DATA")
print(f"{'='*60}")

X_train, X_val, y_train, y_val = train_test_split(
    X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balar
)

print(f"Training samples: {len(X_train):,}")
print(f"Validation samples: {len(X_val):,}")
print(f"Train/Val split: 80/20")
```

```
============================================================
SPLITTING DATA
============================================================
Training samples: 17,497
Validation samples: 4,375
Train/Val split: 80/20
```

Build CNN Model Architecture

Model Architecture Overview

Our CNN has **4 convolutional blocks** that progressively learn more complex features:

- Block 1 (32 filters)
- Block 2 (64 filters)
- Block 3 (128 filters)
- Block 4 (256 filters)
- Classifier

Key Design Choices

Why Global Average Pooling instead of Flatten?

- Our vehicles are small (64×64 pixels)
- Global Average Pooling reduces overfitting for small images
- Works better for small object classification
- Reduces total parameters (fewer weights to train)

Why Batch Normalization?

- Helps training go faster
- Makes model more stable
- Reduces chance of getting stuck during training

Why Dropout (0.3)?

- Prevents overfitting (memorizing training data)
- Forces model to learn general patterns, not specific examples
- 30% of neurons randomly turned off during training

Why Adam Optimizer?

- Works well for most problems without manual tuning
- Adjusts learning rate automatically for each parameter
- Fast convergence (reaches good accuracy quickly)

In [ ]:
```
# Build CNN Model (Optimized Architecture)

def create_vehicle_classifier(input_shape=(64, 64, 3), num_classes=4):
    """
    Optimized CNN architecture for vehicle classification
    Similar to the PyTorch example but adapted for TensorFlow
    """
    model = models.Sequential([
        # Input layer
        layers.Input(shape=input_shape),
```

```
        # Block 1: Initial feature extraction
        layers.Conv2D(32, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),

        # Block 2: Mid-level features
        layers.Conv2D(64, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),

        # Block 3: High-level features
        layers.Conv2D(128, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),

        # Block 4: Abstract features
        layers.Conv2D(256, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.GlobalAveragePooling2D(),  # Better than flatten for small ob

        # Classifier head
        layers.Dropout(0.3),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(num_classes, activation='softmax')
    ])

    return model

print(f"\n{'='*60}")
print("BUILDING MODEL")
print(f"{'='*60}")

model = create_vehicle_classifier(input_shape=(64, 64, 3), num_classes=num_c

# Compile model WITH class weights
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Print model summary
model.summary()

print(f"\nTotal parameters: {model.count_params():,}")
print(f"Note: Class weights will be applied during training to handle '{minc
```

```
============================================================
BUILDING MODEL
============================================================
```

```
WARNING: All log messages before absl::InitializeLog() is called are written
to STDERR
I0000 00:00:1764824691.080819    6072 gpu_device.cc:2020] Created device /jo
b:localhost/replica:0/task:0/device:GPU:0 with 4080 MB memory:  -> device:
0, name: NVIDIA GeForce GTX 1660 Ti, pci bus id: 0000:01:00.0, compute capab
ility: 7.5
```

Model: "sequential"

Layer (type)	Output Shape	Par
conv2d (Conv2D)	(None, 64, 64, 32)	
batch_normalization (BatchNormalization)	(None, 64, 64, 32)	
activation (Activation)	(None, 64, 64, 32)	
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	
activation_1 (Activation)	(None, 32, 32, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	
activation_2 (Activation)	(None, 16, 16, 128)	
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	
conv2d_3 (Conv2D)	(None, 8, 8, 256)	295
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 256)	1
activation_3 (Activation)	(None, 8, 8, 256)	
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	
dropout (Dropout)	(None, 256)	
dense (Dense)	(None, 128)	32
dropout_1 (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 4)	

Total params: 423,748 (1.62 MB)

Trainable params: 422,788 (1.61 MB)

Non-trainable params: 960 (3.75 KB)

```
Total parameters: 423,748
Note: Class weights will be applied during training to handle 'others' imbal
ance
```

Training the CNN Model

Training Configuration

- **Batch Size:** 32
- **Epochs:** 20
- **Optimizer:** Adam (learning rate = 0.001)
- **Loss Function:** Sparse Categorical Crossentropy
- **Dataset:** Balanced across all four classes

Callbacks Used During Training

1. Early Stopping

- Monitors **validation accuracy**
- Stops training when accuracy stops improving
- Prevents overfitting and unnecessary extra epochs

2. ReduceLROnPlateau

- Monitors **validation loss**
- Reduces learning rate when progress stagnates
- Helps the model escape local minima

Training Process

We train the model for up to **20 epochs**, but EarlyStopping may reduce this number depending on validation performance.

Final Training Results

Training Summary

- **Total Epochs Trained:** 20
- **Best Validation Accuracy:** 0.9959 (99.59%)
- **Final Training Accuracy:** 0.9986 (99.86%)
- **Final Validation Accuracy:** 0.9957 (99.57%)

These results indicate extremely strong model performance with minimal overfitting.

Classification Report (4 Classes)

Class	Precision	Recall	F1-Score	Support
Car	0.9972	0.9916	0.9944	1424
Van	0.9916	0.9958	0.9937	1424
Others	0.9810	1.0000	0.9904	103
Bus	1.0000	1.0000	1.0000	1424

Overall Accuracy: 0.9959 (99.59%)
Macro Avg F1: 0.9946
Weighted Avg F1: 0.9959

The model performs exceptionally well across all four classes, including the previously underrepresented “others” class.

In [ ]:
```
# Training Configuration
print(f"\n{'='*60}")
print("TRAINING CONFIGURATION")
print(f"{'='*60}")

BATCH_SIZE = 32
EPOCHS = 20

print(f"Batch size: {BATCH_SIZE}")
print(f"Epochs: {EPOCHS}")
print(f"Optimizer: Adam (lr=0.001)")
print(f"Loss: Sparse Categorical Crossentropy")
print(f"Dataset: BALANCED (all classes equal)")
print(f"\nBatches per epoch: {len(X_train) // BATCH_SIZE}")

# Callbacks
callbacks = [
    keras.callbacks.EarlyStopping(
        monitor='val_accuracy',
        patience=7,
        restore_best_weights=True,
        verbose=1
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=2,
        min_lr=1e-6,
        verbose=1
    )
]


# Train Model (WITH CLASS WEIGHTS)
print(f"\n{'='*60}")
print("STARTING TRAINING")
print(f"{'='*60}")
print(f"Using class weights to compensate for imbalance")
```

```
print(f"Please wait...\n")

history = model.fit(
    X_train, y_train,
    batch_size=BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=(X_val, y_val),
    class_weight=class_weights,
    callbacks=callbacks,
    verbose=1
)

print(f"\n{'='*60}")
print("TRAINING COMPLETE")
print(f"{'='*60}")


# Training History Visualization
print("\nGenerating training history plots...")

plt.figure(figsize=(14, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy', linewidth=2
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', linew
plt.title('Model Accuracy Over Time', fontsize=14, fontweight='bold')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss', linewidth=2, marker
plt.plot(history.history['val_loss'], label='Validation Loss', linewidth=2,
plt.title('Model Loss Over Time', fontsize=14, fontweight='bold')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.legend(loc='upper right')
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('training_history.png', dpi=300, bbox_inches='tight')
print("✓ Saved training history plot: training_history.png")
plt.show()


# Evaluation and Metrics
print(f"\n{'='*60}")
print("TRAINING SUMMARY")
print(f"{'='*60}")
print(f"Total Epochs Trained: {len(history.history['accuracy'])}")
print(f"Best Validation Accuracy: {max(history.history['val_accuracy']):.4f}
print(f"Final Training Accuracy: {history.history['accuracy'][-1]:.4f} ({his
print(f"Final Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}
```

```python
print(f"{'='*60}")


# Classification Report
print(f"\n{'='*60}")
print("GENERATING CLASSIFICATION REPORT")
print(f"{'='*60}")

# Predict on validation set
y_pred_probs = model.predict(X_val, batch_size=BATCH_SIZE, verbose=0)
y_pred = np.argmax(y_pred_probs, axis=1)

# Class Classification Report
class_names = [idx_to_class[i] for i in range(num_classes)]

print("\n" + "="*60)
print("CLASSIFICATION REPORT - 4 CLASSES")
print("="*60)
print(classification_report(y_val, y_pred, target_names=class_names, digits=
```

```
============================================================
TRAINING CONFIGURATION
============================================================
Batch size: 32
Epochs: 20
Optimizer: Adam (lr=0.001)
Loss: Sparse Categorical Crossentropy
Dataset: BALANCED (all classes equal)

Batches per epoch: 546

============================================================
STARTING TRAINING
============================================================
Expected time: ~5-7 minutes on GTX 1660 Ti
Using class weights to compensate for imbalance
Please wait...
```

```
2025-12-04 00:05:02.276758: W external/local_xla/xla/tsl/framework/cpu_alloc
ator_impl.cc:84] Allocation of 860012544 exceeds 10% of free system memory.
2025-12-04 00:05:05.299827: W external/local_xla/xla/tsl/framework/cpu_alloc
ator_impl.cc:84] Allocation of 860012544 exceeds 10% of free system memory.
```

```
Epoch 1/20
```

```
2025-12-04 00:05:10.178095: I external/local_xla/xla/service/service.cc:163] XLA service 0x757b38037f80 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
2025-12-04 00:05:10.178147: I external/local_xla/xla/service/service.cc:171]   StreamExecutor device (0): NVIDIA GeForce GTX 1660 Ti, Compute Capability 7.5
2025-12-04 00:05:10.361085: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2025-12-04 00:05:11.105324: I external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:473] Loaded cuDNN version 91002
2025-12-04 00:05:12.471475: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[32,32,64,64]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,3,64,64]{3,2,1,0}, f32[32,3,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:05:12.597245: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[32,64,32,32]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,32,32,32]{3,2,1,0}, f32[64,32,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:05:12.774699: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[32,128,16,16]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,64,16,16]{3,2,1,0}, f32[128,64,3,3]{3,2,1,0}, f32[128]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:05:12.851967: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[32,256,8,8]{3,2,1,0}, u8[0]{0}) custom-call(f32[32,128,8,8]{3,2,1,0}, f32[256,128,3,3]{3,2,1,0}, f32[256]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
```

12/547 ━━━━━━━━━━━━━━━━━━━━ **7s** 14ms/step - accuracy: 0.3720 - loss: 1.2816

```
I0000 00:00:1764824716.885544    7257 device_compiler.h:196] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
```

545/547 ━━━━━━━━━━━━━━━━━━━━ **0s** 10ms/step - accuracy: 0.7086 - loss: 0.6629

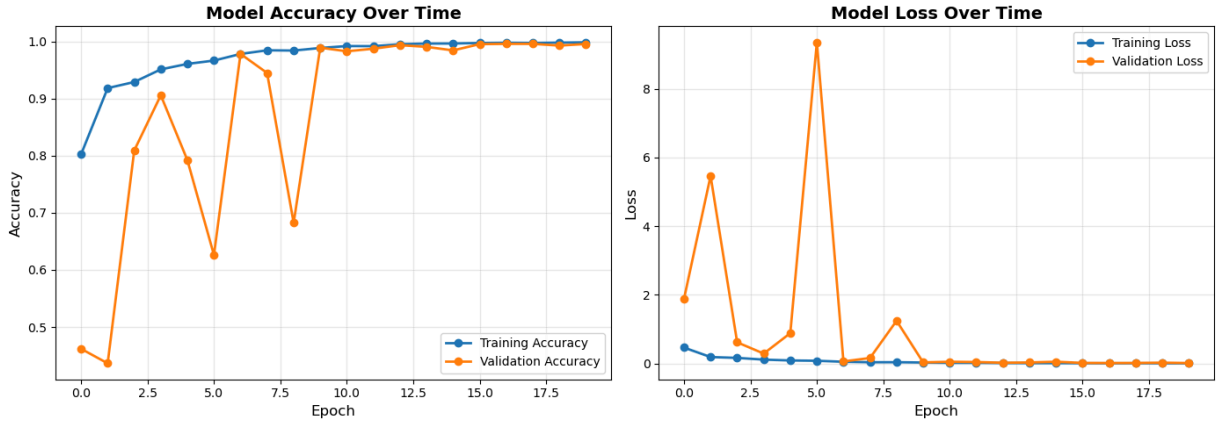
```
2025-12-04 00:05:23.314575: I external/local_xla/xla/service/gpu/autotuning/
conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=
2} for conv (f32[25,32,64,64]{3,2,1,0}, u8[0]{0}) custom-call(f32[25,3,64,6
4]{3,2,1,0}, f32[32,3,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3 pad=1_1x1
_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivat
ionForward", backend_config={"operation_queue_id":"0","wait_on_operation_que
ues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_
scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedul
e":false,"reification_cost":[]}
2025-12-04 00:05:23.341385: I external/local_xla/xla/service/gpu/autotuning/
conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=
2} for conv (f32[25,64,32,32]{3,2,1,0}, u8[0]{0}) custom-call(f32[25,32,32,3
2]{3,2,1,0}, f32[64,32,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3 pad=1_1x
1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActiva
tionForward", backend_config={"operation_queue_id":"0","wait_on_operation_qu
eues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result
_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedul
e":false,"reification_cost":[]}
2025-12-04 00:05:23.420454: I external/local_xla/xla/service/gpu/autotuning/
conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=
2} for conv (f32[25,128,16,16]{3,2,1,0}, u8[0]{0}) custom-call(f32[25,64,16,
16]{3,2,1,0}, f32[128,64,3,3]{3,2,1,0}, f32[128]{0}), window={size=3x3 pad=1
_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasAct
ivationForward", backend_config={"operation_queue_id":"0","wait_on_operation
_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_res
ult_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_sched
ule":false,"reification_cost":[]}
2025-12-04 00:05:23.498060: I external/local_xla/xla/service/gpu/autotuning/
conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=
2} for conv (f32[25,256,8,8]{3,2,1,0}, u8[0]{0}) custom-call(f32[25,128,8,8]
{3,2,1,0}, f32[256,128,3,3]{3,2,1,0}, f32[256]{0}), window={size=3x3 pad=1_1
x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActiv
ationForward", backend_config={"operation_queue_id":"0","wait_on_operation_q
ueues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_resul
t_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedul
e":false,"reification_cost":[]}
```

547/547 ━━━━━━━━━━━━━━━━━━━━ **0s** 19ms/step - accuracy: 0.7090 - loss: 0.6621

```
2025-12-04 00:05:27.682851: W external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:84] Allocation of 215040000 exceeds 10% of free system memory.
2025-12-04 00:05:28.220551: W external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:84] Allocation of 215040000 exceeds 10% of free system memory.
2025-12-04 00:05:29.994720: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[23,32,64,64]{3,2,1,0}, u8[0]{0}) custom-call(f32[23,3,64,64]{3,2,1,0}, f32[32,3,3,3]{3,2,1,0}, f32[32]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:05:30.056798: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[23,64,32,32]{3,2,1,0}, u8[0]{0}) custom-call(f32[23,32,32,32]{3,2,1,0}, f32[64,32,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:05:30.155933: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[23,128,16,16]{3,2,1,0}, u8[0]{0}) custom-call(f32[23,64,16,16]{3,2,1,0}, f32[128,64,3,3]{3,2,1,0}, f32[128]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:05:30.246485: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[23,256,8,8]{3,2,1,0}, u8[0]{0}) custom-call(f32[23,128,8,8]{3,2,1,0}, f32[256,128,3,3]{3,2,1,0}, f32[256]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
```

```
547/547 ━━━━━━━━━━━━━━━━━━━━ 25s 25ms/step - accuracy: 0.7092 - loss: 0.6618
- val_accuracy: 0.4617 - val_loss: 1.8832 - learning_rate: 0.0010
Epoch 2/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9114 - loss: 0.1911
- val_accuracy: 0.4363 - val_loss: 5.4635 - learning_rate: 0.0010
Epoch 3/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9140 - loss: 0.1984
- val_accuracy: 0.8091 - val_loss: 0.6162 - learning_rate: 0.0010
Epoch 4/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.9473 - loss: 0.1215
- val_accuracy: 0.9058 - val_loss: 0.2881 - learning_rate: 0.0010
Epoch 5/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.9520 - loss: 0.1084
- val_accuracy: 0.7922 - val_loss: 0.8824 - learning_rate: 0.0010
Epoch 6/20
545/547 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9638 - loss: 0.0816
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9638 - loss: 0.0815
- val_accuracy: 0.6261 - val_loss: 9.3554 - learning_rate: 0.0010
Epoch 7/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9689 - loss: 0.0703
- val_accuracy: 0.9783 - val_loss: 0.0587 - learning_rate: 5.0000e-04
Epoch 8/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.9832 - loss: 0.0415
- val_accuracy: 0.9449 - val_loss: 0.1596 - learning_rate: 5.0000e-04
Epoch 9/20
545/547 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9848 - loss: 0.0333
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9848 - loss: 0.0334
- val_accuracy: 0.6823 - val_loss: 1.2434 - learning_rate: 5.0000e-04
Epoch 10/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9843 - loss: 0.0401
- val_accuracy: 0.9893 - val_loss: 0.0310 - learning_rate: 2.5000e-04
Epoch 11/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9904 - loss: 0.0246
- val_accuracy: 0.9829 - val_loss: 0.0498 - learning_rate: 2.5000e-04
Epoch 12/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9893 - loss: 0.0247
Epoch 12: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9893 - loss: 0.0247
- val_accuracy: 0.9874 - val_loss: 0.0414 - learning_rate: 2.5000e-04
Epoch 13/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - accuracy: 0.9935 - loss: 0.0177
- val_accuracy: 0.9936 - val_loss: 0.0210 - learning_rate: 1.2500e-04
Epoch 14/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9959 - loss: 0.0135
- val_accuracy: 0.9906 - val_loss: 0.0288 - learning_rate: 1.2500e-04
Epoch 15/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9962 - loss: 0.0125
Epoch 15: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9962 - loss: 0.0125
- val_accuracy: 0.9845 - val_loss: 0.0508 - learning_rate: 1.2500e-04
Epoch 16/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9963 - loss: 0.0121
- val_accuracy: 0.9954 - val_loss: 0.0153 - learning_rate: 6.2500e-05
Epoch 17/20
```

```
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9973 - loss: 0.0081
- val_accuracy: 0.9959 - val_loss: 0.0135 - learning_rate: 6.2500e-05
Epoch 18/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9967 - loss: 0.0089
- val_accuracy: 0.9959 - val_loss: 0.0146 - learning_rate: 6.2500e-05
Epoch 19/20
542/547 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9972 - loss: 0.0084
Epoch 19: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9972 - loss: 0.0083
- val_accuracy: 0.9929 - val_loss: 0.0229 - learning_rate: 6.2500e-05
Epoch 20/20
547/547 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9982 - loss: 0.0066
- val_accuracy: 0.9957 - val_loss: 0.0124 - learning_rate: 3.1250e-05
Restoring model weights from the end of the best epoch: 17.

============================================================
TRAINING COMPLETE
============================================================

Generating training history plots...
✓ Saved training history plot: training_history.png
```




```
============================================================
TRAINING SUMMARY
============================================================
Total Epochs Trained: 20
Best Validation Accuracy: 0.9959 (99.59%)
Final Training Accuracy: 0.9986 (99.86%)
Final Validation Accuracy: 0.9957 (99.57%)
============================================================

============================================================
GENERATING CLASSIFICATION REPORT
============================================================
2025-12-04 00:07:34.334776: W external/local_xla/xla/tsl/framework/cpu_alloc
ator_impl.cc:84] Allocation of 215040000 exceeds 10% of free system memory.
```

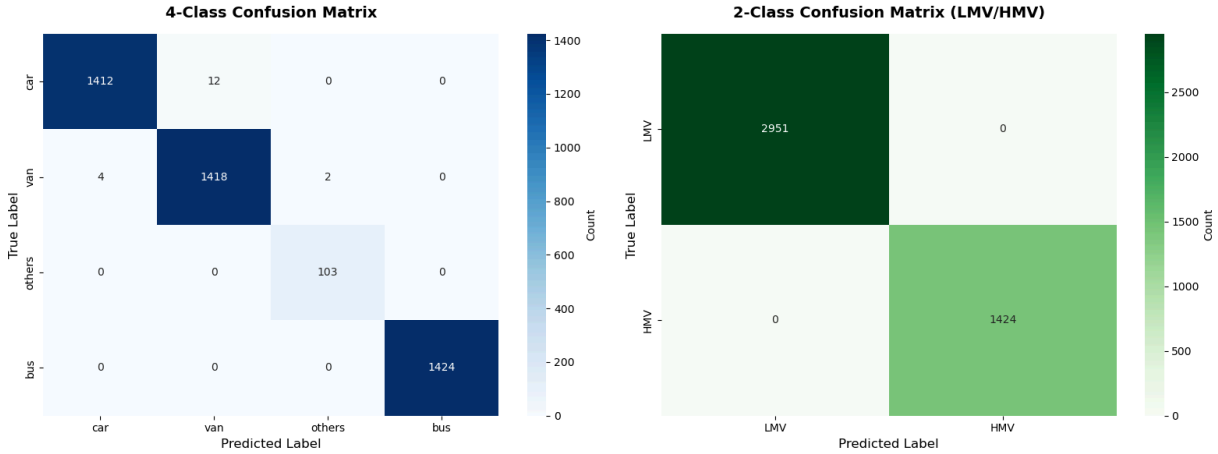
```
============================================================
CLASSIFICATION REPORT - 4 CLASSES
============================================================
              precision    recall  f1-score   support

         car     0.9972    0.9916    0.9944      1424
         van     0.9916    0.9958    0.9937      1424
      others     0.9810    1.0000    0.9904       103
         bus     1.0000    1.0000    1.0000      1424

    accuracy                         0.9959      4375
   macro avg     0.9924    0.9968    0.9946      4375
weighted avg     0.9959    0.9959    0.9959      4375
```

In [ ]:

```python
# LMV/HMV Classification

def map_to_lmv_hmv(predictions_4class, class_names):
    """
    Map 4-class predictions to 2-class (LMV/HMV)
    LMV (0): Car, Van, Others
    HMV (1): Bus
    """
    mapping = {}
    for idx, name in enumerate(class_names):
        if name.lower() in ['car', 'van', 'others']:
            mapping[idx] = 0  # LMV
        elif name.lower() in ['bus', 'truck']:
            mapping[idx] = 1  # HMV
        else:
            mapping[idx] = 0  # Default to LMV

    return np.array([mapping[x] for x in predictions_4class])

# Convert to LMV/HMV
y_val_2class = map_to_lmv_hmv(y_val, class_names)
y_pred_2class = map_to_lmv_hmv(y_pred, class_names)

print("\n" + "="*60)
print("CLASSIFICATION REPORT - LMV/HMV")
print("="*60)
print(classification_report(y_val_2class, y_pred_2class,
                            target_names=['LMV (Light)', 'HMV (Heavy)'],
                            digits=4))


# Confusion Matrices

print("\nGenerating confusion matrices...")

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# 4-Class Confusion Matrix
cm_4class = confusion_matrix(y_val, y_pred)
sns.heatmap(cm_4class, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names,
```

```
            ax=axes[0], cbar_kws={'label': 'Count'})
axes[0].set_title('4-Class Confusion Matrix', fontsize=14, fontweight='bold'
axes[0].set_ylabel('True Label', fontsize=12)
axes[0].set_xlabel('Predicted Label', fontsize=12)

# 2-Class Confusion Matrix
cm_2class = confusion_matrix(y_val_2class, y_pred_2class)
sns.heatmap(cm_2class, annot=True, fmt='d', cmap='Greens',
            xticklabels=['LMV', 'HMV'], yticklabels=['LMV', 'HMV'],
            ax=axes[1], cbar_kws={'label': 'Count'})
axes[1].set_title('2-Class Confusion Matrix (LMV/HMV)', fontsize=14, fontwei
axes[1].set_ylabel('True Label', fontsize=12)
axes[1].set_xlabel('Predicted Label', fontsize=12)

plt.tight_layout()
plt.show()
```

```
============================================================
CLASSIFICATION REPORT - LMV/HMV
============================================================
              precision    recall  f1-score   support

 LMV (Light)     1.0000    1.0000    1.0000      2951
 HMV (Heavy)     1.0000    1.0000    1.0000      1424

    accuracy                         1.0000      4375
   macro avg     1.0000    1.0000    1.0000      4375
weighted avg     1.0000    1.0000    1.0000      4375


Generating confusion matrices...
✓ Saved confusion matrices: confusion_matrices.png
```



Detect Vehicles in Full Frames (LMV vs HMV Classification)

Classifying LMV - HMV

full-size traffic frames from the UA-DETRAC dataset, reads their bounding-box annotations, extracts each vehicle, classifies it using our trained CNN model, and overlays **colored bounding boxes** for:

- **LMV (Light Motor Vehicles)** – Cars, Vans, Others
- **HMV (Heavy Motor Vehicles)** – Buses (and Trucks, if present)

Each detected vehicle gets:

- A bounding box (green = LMV, red = HMV)
- A label with class name + confidence
- A final count of LMV vs HMV in the frame

Class Mapping Logic (LMV vs HMV)

Our main classifier outputs **4 classes**:

- Car
- Van
- Others
- Bus

These are mapped into two high-level traffic groups:

4-Class Category	Mapped Class	Group Name
Car	0	LMV
Van	0	LMV
Others	0	LMV
Bus	1	HMV

- Car , Van , Other = LMV
- Bus = HMV

In [ ]:
```python
# Detect Vehicles in Full Frames with Bounding Boxes
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
import cv2
from pathlib import Path
import xml.etree.ElementTree as ET

# Define colors and class names
colors_2class = {0: 'green', 1: 'red'}
class_names_2 = {0: 'LMV (Light)', 1: 'HMV (Heavy)'}

# Map to LMV/HMV (using your exact function from training)
def map_to_lmv_hmv(predictions_4class, class_names):
    """
    Map 4-class predictions to 2-class (LMV/HMV)
    LMV (0): Car, Van, Others
    HMV (1): Bus
    """
```

```
    mapping = {}
    for idx, name in enumerate(class_names):
        if name.lower() in ['car', 'van', 'others']:
            mapping[idx] = 0  # LMV
        elif name.lower() in ['bus', 'truck']:
            mapping[idx] = 1  # HMV
        else:
            mapping[idx] = 0  # Default to LMV

    return np.array([mapping[x] for x in predictions_4class])

# Parse XML annotations
def parse_detrac_xml(xml_path):
    """Parse DETRAC XML file and return frame-level annotations"""
    tree = ET.parse(str(xml_path))
    root = tree.getroot()

    annotations = {}
    for frame in root.findall('.//frame'):
        frame_num = int(frame.attrib['num'])
        vehicles = []

        target_list = frame.find('target_list')
        if target_list is not None:
            for target in target_list.findall('target'):
                box = target.find('box')
                attr = target.find('attribute')

                if box is not None:
                    vehicle = {
                        'bbox': {
                            'left': float(box.attrib['left']),
                            'top': float(box.attrib['top']),
                            'width': float(box.attrib['width']),
                            'height': float(box.attrib['height'])
                        },
                        'label': attr.attrib.get('vehicle_type', 'unknown').
                    }
                    vehicles.append(vehicle)

        annotations[frame_num] = vehicles

    return annotations

# Detect vehicles in a single frame
def detect_vehicles_in_frame(seq_name, frame_num):
    """
    Load a full frame and detect all vehicles in it
    """
    # Load frame
    img_path = IMAGES_PATH / seq_name / f"img{frame_num:05d}.jpg"
    if not img_path.exists():
        print(f"Frame not found: {img_path}")
        return

    frame = cv2.imread(str(img_path))
```

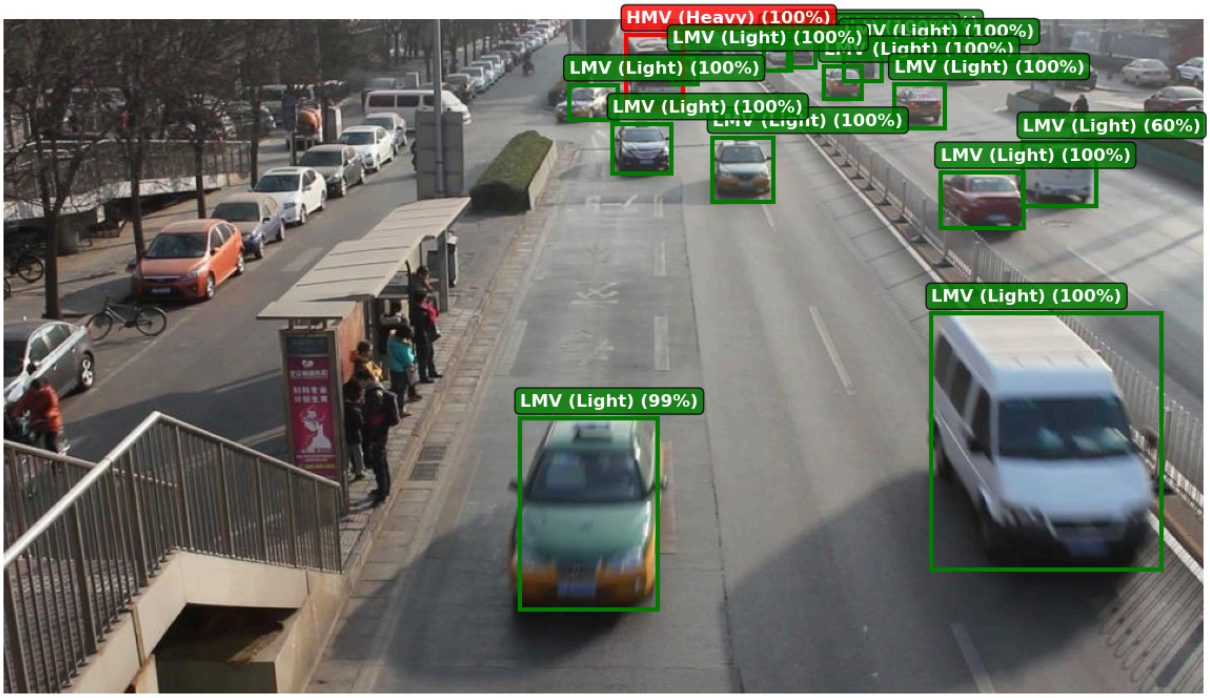
```
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Load annotations
    xml_path = TRAIN_ANNOT_PATH / f"{seq_name}.xml"

    if not xml_path.exists():
        print(f"Annotations not found for {seq_name}")
        return

    annotations = parse_detrac_xml(xml_path)
    vehicles = annotations.get(frame_num, [])

    if len(vehicles) == 0:
        print(f"No vehicles in this frame")
        return

    print(f"Found {len(vehicles)} vehicles in frame {frame_num}")

    # Prepare for plotting
    fig, ax = plt.subplots(1, 1, figsize=(12, 10))
    ax.imshow(frame)

    # Process each vehicle
    detected_count = 0
    lmv_count = 0
    hmv_count = 0

    for vehicle in vehicles:
        bbox = vehicle['bbox']

        # Crop vehicle
        left = int(max(0, bbox['left']))
        top = int(max(0, bbox['top']))
        right = int(min(frame.shape[1], bbox['left'] + bbox['width']))
        bottom = int(min(frame.shape[0], bbox['top'] + bbox['height']))

        if right <= left or bottom <= top:
            continue

        vehicle_crop = frame[top:bottom, left:right]

        # Skip if crop is too small
        if vehicle_crop.shape[0] < 20 or vehicle_crop.shape[1] < 20:
            continue

        # Resize and predict (use 64x64 to match training)
        vehicle_resized = cv2.resize(vehicle_crop, (64, 64)) / 255.0
        vehicle_input = np.expand_dims(vehicle_resized, axis=0)

        pred = model.predict(vehicle_input, verbose=0)
        pred_class_4 = np.argmax(pred[0])
        confidence = np.max(pred[0])

        # Map to LMV/HMV
        pred_lmv_hmv = map_to_lmv_hmv([pred_class_4], class_names)[0]
```

```
        # Count vehicles
        if pred_lmv_hmv == 0:
            lmv_count += 1
        else:
            hmv_count += 1

        # Draw bounding box
        rect = patches.Rectangle(
            (left, top), right - left, bottom - top,
            linewidth=3,
            edgecolor=colors_2class[pred_lmv_hmv],
            facecolor='none'
        )
        ax.add_patch(rect)

        # Add label
        label_text = f"{class_names_2[pred_lmv_hmv]} ({confidence:.0%})"
        ax.text(left, top - 10, label_text,
                bbox=dict(boxstyle='round', facecolor=colors_2class[pred_lmv
                fontsize=12, color='white', fontweight='bold')

        detected_count += 1

    ax.axis('off')
    title = f'Frame Detection: {seq_name} - Frame {frame_num}\n'
    title += f'{detected_count} vehicles (LMV: {lmv_count}, HMV: {hmv_count}
    ax.set_title(title, fontsize=16, fontweight='bold', pad=20)
    plt.tight_layout()
    print(f"✓ Detected {detected_count} vehicles (LMV: {lmv_count}, HMV: {hm
    plt.show()

# Example: Detect vehicles in 3 different frames
print("\nDetecting vehicles in sample frames...")
detect_vehicles_in_frame('MVI_20011', 570)
detect_vehicles_in_frame('MVI_39761', 26)
detect_vehicles_in_frame('MVI_20034', 150)
```

```
Detecting vehicles in sample frames...
Found 14 vehicles in frame 570
```

2025-12-04 00:08:13.666293: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[1,64,32,32]{3,2,1,0}, u8[0]{0}) custom-call(f32[1,32,32,32]{3,2,1,0}, f32[64,32,3,3]{3,2,1,0}, f32[64]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:08:13.725078: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[1,128,16,16]{3,2,1,0}, u8[0]{0}) custom-call(f32[1,64,16,16]{3,2,1,0}, f32[128,64,3,3]{3,2,1,0}, f32[128]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
2025-12-04 00:08:13.800342: I external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:546] Omitted potentially buggy algorithm eng14{k25=2} for conv (f32[1,256,8,8]{3,2,1,0}, u8[0]{0}) custom-call(f32[1,128,8,8]{3,2,1,0}, f32[256,128,3,3]{3,2,1,0}, f32[256]{0}), window={size=3x3 pad=1_1x1_1}, dim_labels=bf01_oi01->bf01, custom_call_target="__cudnn$convBiasActivationForward", backend_config={"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_config":{"activation_mode":"kNone","conv_result_scale":1,"side_input_scale":0,"leakyrelu_alpha":0},"force_earliest_schedule":false,"reification_cost":[]}
✓ Detected 14 vehicles (LMV: 13, HMV: 1)
✓ Saved as 'frame_detection_MVI_20011_570.png'

Frame Detection: MVI_20011 - Frame 570
14 vehicles (LMV: 13, HMV: 1)

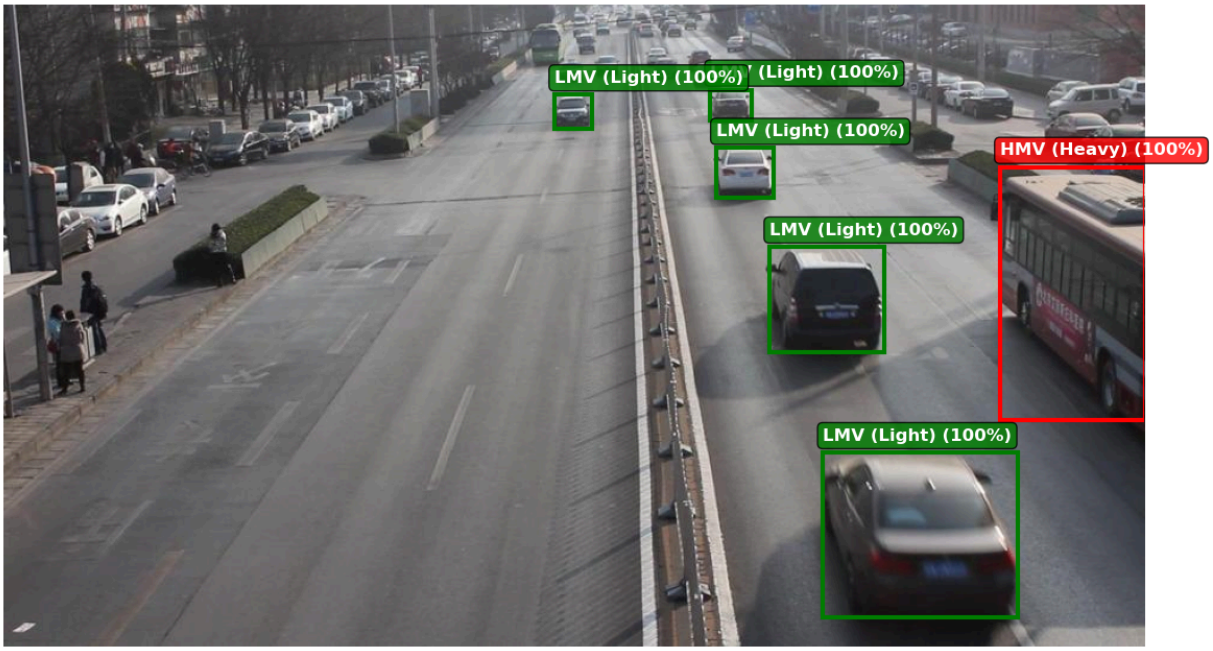


Found 5 vehicles in frame 26
✓ Detected 5 vehicles (LMV: 5, HMV: 0)
✓ Saved as 'frame_detection_MVI_39761_26.png'

Frame Detection: MVI_39761 - Frame 26
5 vehicles (LMV: 5, HMV: 0)


```
Found 6 vehicles in frame 150
✓ Detected 6 vehicles (LMV: 5, HMV: 1)
✓ Saved as 'frame_detection_MVI_20034_150.png'
```

Frame Detection: MVI_20034 - Frame 150
6 vehicles (LMV: 5, HMV: 1)



Video Vehicle Detection and Tracking Pipeline

---

link to the video used for the detection - https://www.pexels.com/video/a-multi-lane-highway-built-in-the-counryside-5473765/

This pipeline processes a video to detect, classify, and track vehicles (LMV/HMV) with **stable bounding boxes** using YOLOv8 and a custom CNN classifier. It eliminates flickering of bounding boxes by maintaining tracking across frames.

> This section requires the `ultralytics` package and is **computationally heavy**. It is purely experimental and not part of the final evaluation.

Features

- Detects vehicles in each frame using **YOLOv8**.
- Classifies vehicles as **LMV (Light Motor Vehicle)** or **HMV (Heavy Motor Vehicle)** using a **CNN model**.
- Tracks vehicles across frames with **ID assignment**.
- Applies **size-based filtering** to reduce false positives for HMV.
- Displays **stable bounding boxes** with vehicle ID, class, and confidence.

Functions

1. Tracking Helper Functions

- reset_tracker(): Resets the tracker for a new video.
- calculate_centroid(bbox): Returns the centroid of a bounding box [x, y, w, h].
- calculate_distance(point1, point2): Computes Euclidean distance between two points.
- get_smoothed_class(class_history): Returns the majority class from past frames.
- update_tracker(detections): Updates tracked objects using nearest neighbor matching and smooths class predictions.

2. YOLO and CNN Classification

- load_yolo8_model(model_path): Loads the YOLOv8 model.
- detect_vehicles_yolo8(frame, yolo_model, confidence_threshold): Detects vehicles using YOLOv8.
- map_to_lmv_hmv(predictions_4class, class_names): Maps 4-class CNN predictions to LMV/HMV.
- classify_vehicle_with_size_filter(vehicle_crop, bbox_width, bbox_height, frame_height): Classifies vehicle crops and applies a size filter to prevent false HMV detections.

Pipeline Steps:

1. Reset tracker.
2. Load YOLOv8 model.
3. Open input video and initialize output writer.
4. For each frame:
    - Detect vehicles with YOLOv8.

- Crop and classify vehicles with CNN.
- Update tracker for stable IDs and bounding boxes.
- Draw boxes, labels, and frame statistics.
- Write processed frame to output video.

In [35]:

```python
# COMPLETE VIDEO PROCESSING WITH TRACKING

import cv2
import numpy as np
from pathlib import Path
from ultralytics import YOLO


# Global Variables for Tracking
tracker_objects = {}
tracker_next_id = 0
tracker_max_disappeared = 10
tracker_max_distance = 150
tracker_history_length = 5

# Colors and class names
colors_2class = {0: (0, 255, 0), 1: (0, 0, 255)}  # BGR: Green=LMV, Red=HMV
class_names_2 = {0: 'LMV', 1: 'HMV'}


# Tracking Helper Functions
# Reset tracker state (call at start of new video)
def reset_tracker():

    global tracker_objects, tracker_next_id
    tracker_objects = {}
    tracker_next_id = 0
    print("Tracker reset")

# Calculate centroid from bounding box [x, y, w, h]
def calculate_centroid(bbox):

    x, y, w, h = bbox
    cx = x + w // 2
    cy = y + h // 2
    return (cx, cy)

# Calculate Euclidean distance between two points
def calculate_distance(point1, point2):

    return np.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)

# Get most common class from history (majority vote)
def get_smoothed_class(class_history):

    if not class_history:
        return None
    return max(set(class_history), key=class_history.count)
```

```python
# Update tracker with new detections
def update_tracker(detections):

    global tracker_objects, tracker_next_id

    # If no detections, increment disappeared counter
    if len(detections) == 0:
        to_delete = []
        for obj_id in tracker_objects.keys():
            tracker_objects[obj_id]['disappeared'] += 1
            if tracker_objects[obj_id]['disappeared'] > tracker_max_disappea
                to_delete.append(obj_id)

        for obj_id in to_delete:
            del tracker_objects[obj_id]

        return tracker_objects

    # Calculate centroids for new detections
    detection_centroids = []
    for (bbox, vehicle_class, confidence) in detections:
        centroid = calculate_centroid(bbox)
        detection_centroids.append(centroid)

    # If no existing tracked objects, register all detections
    if len(tracker_objects) == 0:
        for i, (bbox, vehicle_class, confidence) in enumerate(detections):
            centroid = detection_centroids[i]
            tracker_objects[tracker_next_id] = {
                'centroid': centroid,
                'bbox': bbox,
                'class': vehicle_class,
                'conf': confidence,
                'disappeared': 0,
                'history': [vehicle_class]
            }
            tracker_next_id += 1
        return tracker_objects

    # Match existing objects to new detections
    object_ids = list(tracker_objects.keys())
    object_centroids = [tracker_objects[oid]['centroid'] for oid in object_i

    # Calculate distance matrix
    num_objects = len(object_centroids)
    num_detections = len(detection_centroids)
    distances = np.zeros((num_objects, num_detections))

    for i, obj_centroid in enumerate(object_centroids):
        for j, det_centroid in enumerate(detection_centroids):
            distances[i, j] = calculate_distance(obj_centroid, det_centroid)

    # Match using greedy nearest neighbor
    rows = distances.min(axis=1).argsort()
    cols = distances.argmin(axis=1)[rows]
```

```
    used_rows = set()
    used_cols = set()

    # Update matched objects
    for row, col in zip(rows, cols):
        if row in used_rows or col in used_cols:
            continue

        if distances[row, col] > tracker_max_distance:
            continue

        obj_id = object_ids[row]
        centroid = detection_centroids[col]
        bbox, vehicle_class, confidence = detections[col]

        # Update class history for smoothing
        tracker_objects[obj_id]['history'].append(vehicle_class)
        if len(tracker_objects[obj_id]['history']) > tracker_history_length:
            tracker_objects[obj_id]['history'].pop(0)

        # Get smoothed class (majority vote)
        smoothed_class = get_smoothed_class(tracker_objects[obj_id]['history

        # Update object
        tracker_objects[obj_id]['centroid'] = centroid
        tracker_objects[obj_id]['bbox'] = bbox
        tracker_objects[obj_id]['class'] = smoothed_class
        tracker_objects[obj_id]['conf'] = confidence
        tracker_objects[obj_id]['disappeared'] = 0

        used_rows.add(row)
        used_cols.add(col)

    # Register new detections (unmatched)
    for col in range(num_detections):
        if col in used_cols:
            continue

        centroid = detection_centroids[col]
        bbox, vehicle_class, confidence = detections[col]

        tracker_objects[tracker_next_id] = {
            'centroid': centroid,
            'bbox': bbox,
            'class': vehicle_class,
            'conf': confidence,
            'disappeared': 0,
            'history': [vehicle_class]
        }
        tracker_next_id += 1

    # Mark disappeared objects (unmatched)
    to_delete = []
    for row in range(num_objects):
        if row in used_rows:
            continue
```

```
        obj_id = object_ids[row]
        tracker_objects[obj_id]['disappeared'] += 1

        if tracker_objects[obj_id]['disappeared'] > tracker_max_disappeared:
            to_delete.append(obj_id)

    # Remove disappeared objects
    for obj_id in to_delete:
        del tracker_objects[obj_id]

    return tracker_objects


# YOLO and Classification Functions

def load_yolo8_model(model_path='yolov8n.pt'):
    print(f"Loading YOLOv8 from: {model_path}...")

    if not Path(model_path).exists():
        raise FileNotFoundError(f"Model file not found: {model_path}")

    yolo_model = YOLO(model_path)
    print(f"✓ YOLOv8 loaded successfully")
    return yolo_model

# Detect vehicles in frame using YOLOv8
def detect_vehicles_yolo8(frame, yolo_model, confidence_threshold=0.5):
    vehicle_classes = [2, 3, 5, 7]
    results = yolo_model(frame, conf=confidence_threshold, verbose=False)
    detected_vehicles = []

    for result in results:
        for box in result.boxes:
            if int(box.cls[0]) in vehicle_classes:
                x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                detected_vehicles.append({
                    'bbox': [int(x1), int(y1), int(x2-x1), int(y2-y1)],
                    'confidence': float(box.conf[0]),
                    'yolo_class': result.names[int(box.cls[0])]
                })
    return detected_vehicles

# Map 4-class predictions to 2-class (LMV/HMV)
def map_to_lmv_hmv(predictions_4class, class_names):
    mapping = {}
    for idx, name in enumerate(class_names):
        if name.lower() in ['car', 'van', 'others']:
            mapping[idx] = 0  # LMV
        elif name.lower() in ['bus', 'truck']:
            mapping[idx] = 1  # HMV
        else:
            mapping[idx] = 0  # Default to LMV

    return np.array([mapping[x] for x in predictions_4class])
```

```python
# Classify vehicle but use bounding box size to filter false HMV detections
def classify_vehicle_with_size_filter(vehicle_crop, bbox_width, bbox_height,
    try:
        # Resize to 64x64 (matches your training)
        vehicle_resized = cv2.resize(vehicle_crop, (64, 64))
        vehicle_rgb = cv2.cvtColor(vehicle_resized, cv2.COLOR_BGR2RGB)
        vehicle_input = np.expand_dims(vehicle_rgb / 255.0, axis=0)

        # Predict with your trained model
        pred = model.predict(vehicle_input, verbose=0)
        pred_class_4 = np.argmax(pred[0])
        confidence = np.max(pred[0])

        # Map to LMV/HMV
        pred_lmv_hmv = map_to_lmv_hmv([pred_class_4], class_names)[0]

        # SIZE-BASED FILTER
        relative_height = bbox_height / frame_height
        bbox_area = bbox_width * bbox_height

        # If classified as HMV, check if bounding box is actually large enou
        if pred_lmv_hmv == 1:  # If predicted HMV
            if relative_height < 0.25 or bbox_area < 80000:
                pred_lmv_hmv = 0
                confidence *= 0.6

        return pred_lmv_hmv, confidence, pred_class_4

    except Exception as e:
        return None, 0, None


# MAIN VIDEO PROCESSING WITH TRACKING
# Process video with vehicle tracking to eliminate flickering
def process_video_with_tracking(video_path, output_path='output_tracked.mp4'
                                model_path='yolov8n.pt',
                                yolo_confidence=0.5,
                                cnn_confidence=0.75,
                                skip_frames=1):
    print(f"\n{'='*60}")
    print("VIDEO PROCESSING - WITH TRACKING (NO FLICKER)")
    print(f"{'='*60}")

    # Reset tracker for new video
    reset_tracker()

    # Load YOLO model
    yolo_model = load_yolo8_model(model_path)

    # Open video
    cap = cv2.VideoCapture(str(video_path))

    if not cap.isOpened():
        print(f"Error: Cannot open video {video_path}")
        return
```

```
    # Get video properties
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    print(f"\nVideo properties:")
    print(f"  Resolution: {width}x{height}")
    print(f"  FPS: {fps}")
    print(f"  Total frames: {total_frames}")
    print(f"  Processing every {skip_frames} frame(s)")

    # Create video writer
    out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'),
                          fps, (width, height))

    frame_num = 0
    lmv_counts_per_frame = []
    hmv_counts_per_frame = []

    print(f"\nProcessing video with tracking...")

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_num += 1

        # Detect and classify vehicles
        detections = []

        if frame_num % skip_frames == 0:
            vehicles = detect_vehicles_yolo8(frame, yolo_model, yolo_confide

            for vehicle in vehicles:
                x, y, w, h = vehicle['bbox']

                if w < 30 or h < 30:
                    continue

                vehicle_crop = frame[y:y+h, x:x+w]

                # Classify with size filter
                lmv_hmv, conf, _ = classify_vehicle_with_size_filter(
                    vehicle_crop, w, h, height
                )

                if lmv_hmv is not None and conf >= cnn_confidence:
                    detections.append(([x, y, w, h], lmv_hmv, conf))

        # Update tracker with detections
        tracked_objects = update_tracker(detections)

        # Draw tracked vehicles (stable boxes!)
        frame_lmv = 0
```

```
        frame_hmv = 0

        for obj_id, obj_data in tracked_objects.items():
            bbox = obj_data['bbox']
            vehicle_class = obj_data['class']
            confidence = obj_data['conf']

            x, y, w, h = bbox

            color = colors_2class[vehicle_class]

            # Draw stable bounding box
            cv2.rectangle(frame, (x, y), (x+w, y+h), color, 3)

            # Add label with vehicle ID
            label = f"ID:{obj_id} {class_names_2[vehicle_class]} {confidence
            cv2.putText(frame, label, (x, y-10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

            # Count
            if vehicle_class == 0:
                frame_lmv += 1
            else:
                frame_hmv += 1

        lmv_counts_per_frame.append(frame_lmv)
        hmv_counts_per_frame.append(frame_hmv)

        # Draw stats overlay
        cv2.rectangle(frame, (0, 0), (350, 120), (0, 0, 0), -1)
        cv2.putText(frame, f"Frame: {frame_num}/{total_frames}",
                    (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
        cv2.putText(frame, f"Tracked Vehicles:",
                    (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
        cv2.putText(frame, f"  LMV: {frame_lmv}",
                    (10, 85), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
        cv2.putText(frame, f"  HMV: {frame_hmv}",
                    (10, 110), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        # Write frame
        out.write(frame)

        # Progress update
        if frame_num % 100 == 0:
            progress = (frame_num / total_frames) * 100
            print(f"  Progress: {frame_num}/{total_frames} ({progress:.1f}%)

    # Cleanup
    cap.release()
    out.release()

    # Calculate statistics
    avg_lmv = np.mean(lmv_counts_per_frame)
    avg_hmv = np.mean(hmv_counts_per_frame)

    print(f"\n{'='*60}")
```

```
    print("VIDEO PROCESSING COMPLETE")
    print(f"{'='*60}")
    print(f"Output saved: {output_path}")
    print(f"Total frames processed: {frame_num}")
    print(f"Average LMV per frame: {avg_lmv:.2f}")
    print(f"Average HMV per frame: {avg_hmv:.2f}")
    print(f"Avg LMV/HMV ratio: {avg_lmv/max(0.1, avg_hmv):.2f}:1")
    print(f"Total unique vehicles tracked: {tracker_next_id}")



# Process your video with tracking (no flicker!)
process_video_with_tracking(
    video_path='highway.mp4',
    output_path='highway_tracked.mp4',
    model_path='yolov8n.pt',
    yolo_confidence=0.5,
    cnn_confidence=0.75,
    skip_frames=1
)
```

```
============================================================
VIDEO PROCESSING - WITH TRACKING (NO FLICKER)
============================================================
Tracker reset
Loading YOLOv8 from: yolov8n.pt...
✓ YOLOv8 loaded successfully

Video properties:
  Resolution: 3840x2160
  FPS: 23
  Total frames: 512
  Processing every 1 frame(s)

Processing video with tracking...
  Progress: 100/512 (19.5%) | Tracked: 9 vehicles
  Progress: 200/512 (39.1%) | Tracked: 5 vehicles
  Progress: 300/512 (58.6%) | Tracked: 6 vehicles
  Progress: 400/512 (78.1%) | Tracked: 6 vehicles
  Progress: 500/512 (97.7%) | Tracked: 5 vehicles

============================================================
VIDEO PROCESSING COMPLETE
============================================================
Output saved: highway_tracked.mp4
Total frames processed: 512
Average LMV per frame: 6.12
Average HMV per frame: 0.13
Avg LMV/HMV ratio: 48.23:1
Total unique vehicles tracked: 62
```

Summary – Why This Notebook Exists

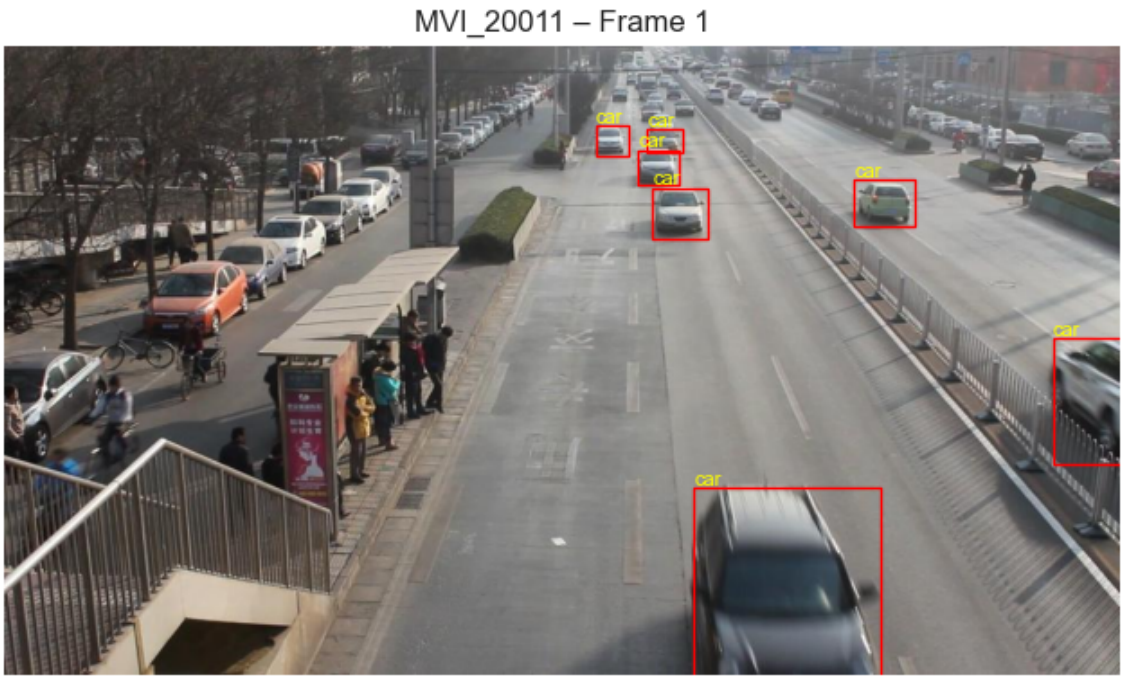
This notebook contains **experimental work** that is *separate* from the other notebooks:

- A **TensorFlow/Keras** version of the vehicle classifier trained on a 20% subset of DETRAC.
- Experiments with **class balancing**, including:
  - Visualizing original vs balanced class distributions,
  - Training the CNN on a more balanced dataset.
- Evaluation of both:
  - Full **multi-class** performance (cars, vans, buses, others),
  - Collapsed **LMV/HMV** performance using 2-class confusion matrices.
- A prototype **YOLO + tracking** pipeline for vehicle detection in real video, which is heavier and not used in the final results.

The main pipeline and project report are based on the **PyTorch** implementation.
This notebook is kept as a **lab notebook / reference** for additional ideas and future extensions.

Appendix B: Figures and Visualizations

Figure 1: Annotated Frame from MVI_20011

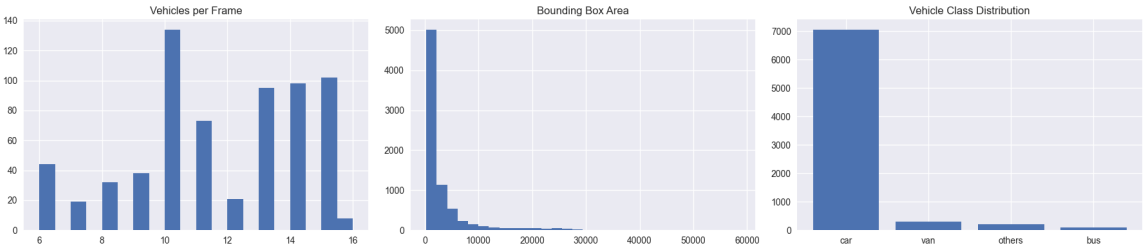


Sample frame from sequence MVI_20011 showing ground-truth bounding box annotations with vehicle class labels and unique identifiers.

Figure 2: Multi-Frame Temporal Visualization

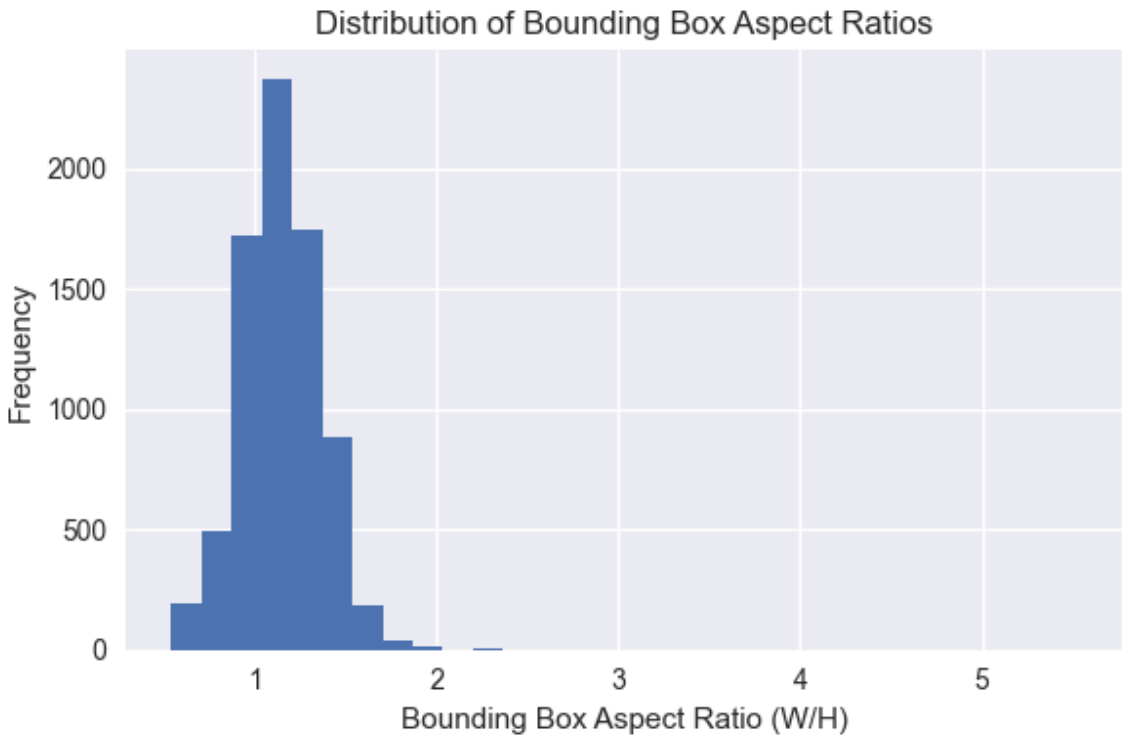
Temporal progression showing frames 1, 10, 25, 50, 75, and 100 with vehicle counts ranging from 6-16 per frame.

Figure 3: Dataset Statistics Summary



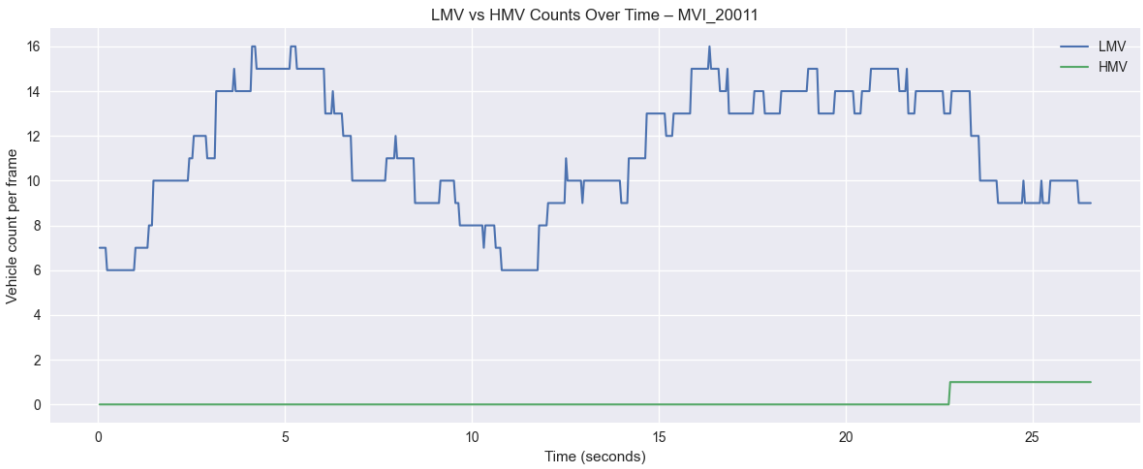
Distribution of vehicles per frame (left), bounding box areas (center), and vehicle class imbalance (right) for MVI_20011.

Figure 4: Bounding Box Aspect Ratio Distribution



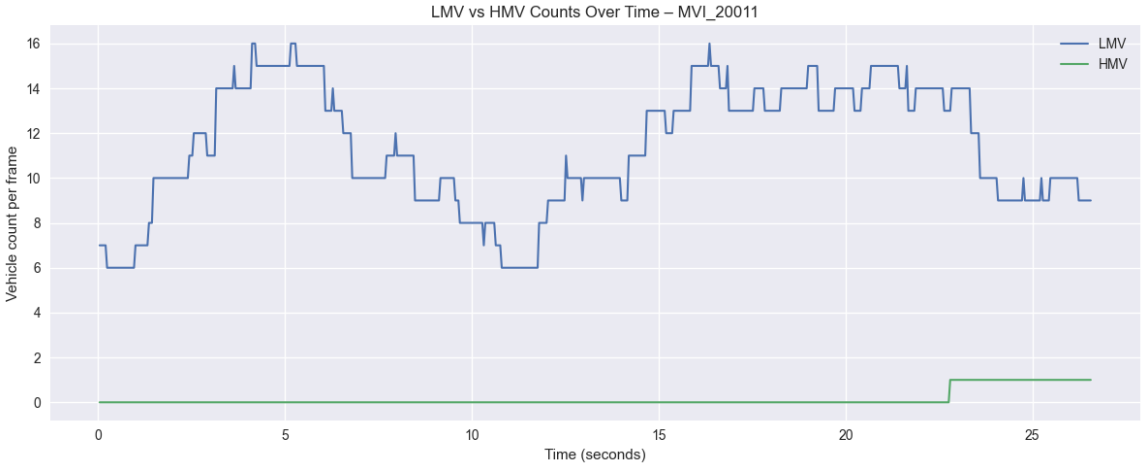
Distribution of aspect ratios across 7,655 vehicle instances showing mean 1.149 (σ=0.225).

Figure 5: Classification Confusion Matrix



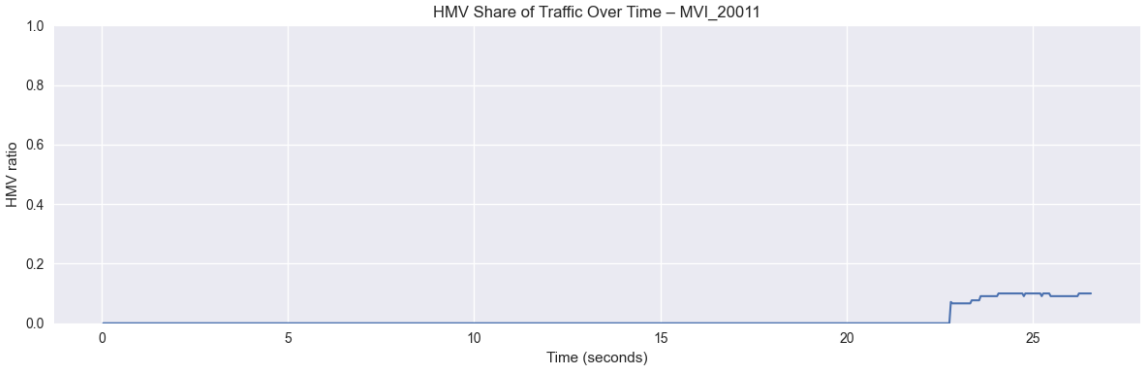
Normalized confusion matrix showing minimal misclassification: 1.5% vans→cars, 1.7% others→cars.

Figure 6: LMV vs HMV Traffic Composition



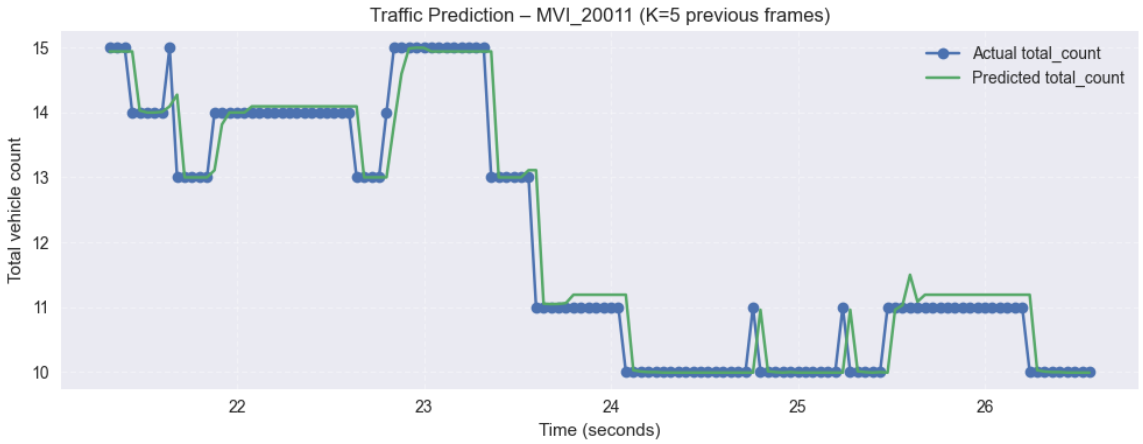
Temporal analysis showing LMV dominance (mean 11.39) vs. sparse HMV occurrences (mean 0.14) over 26.5 seconds.

Figure 7: HMV Share of Traffic Over Time



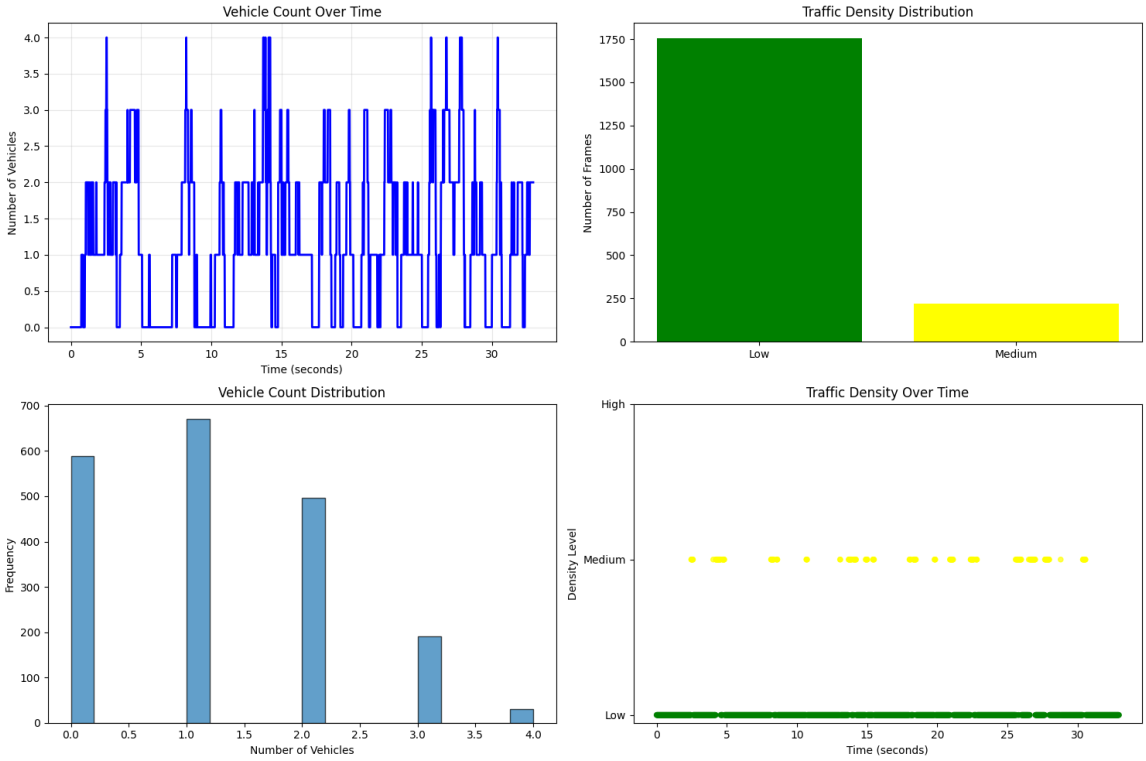
HMV proportion remaining near zero for most duration with spike to ~10% when buses enter (t=23-26s).

Figure 8: Traffic Forecasting Performance



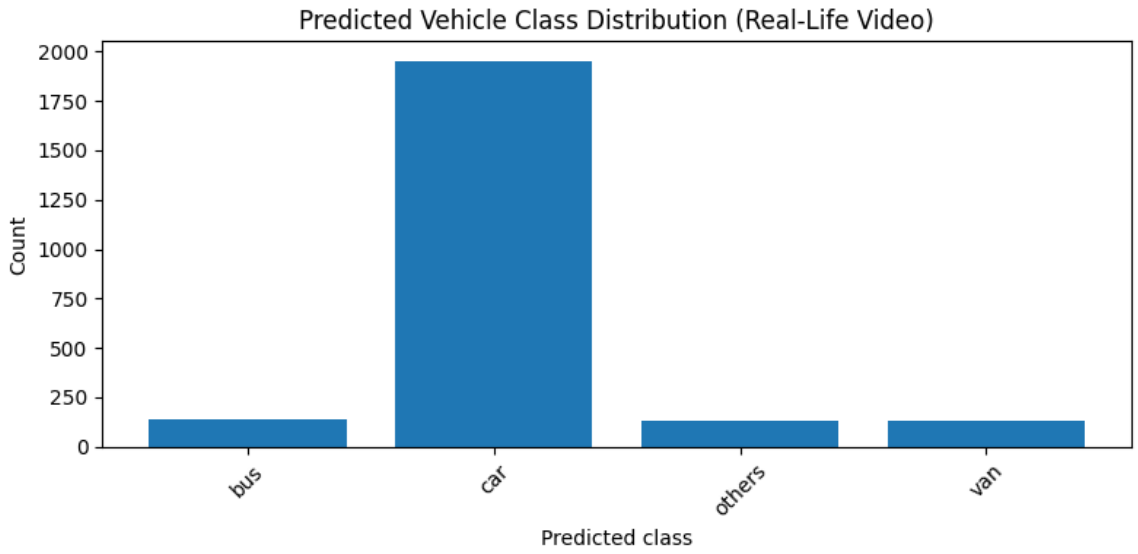
Random Forest predictions (green) vs. actual counts (blue) demonstrating R^2=0.947 and MAE=0.202.

Figure 9: Real-World Video Traffic Analytics



Analysis of traffic_example.mov showing vehicle counts, density distribution, and temporal patterns across 1,977 frames.

Figure 10: Predicted Vehicle Class Distribution

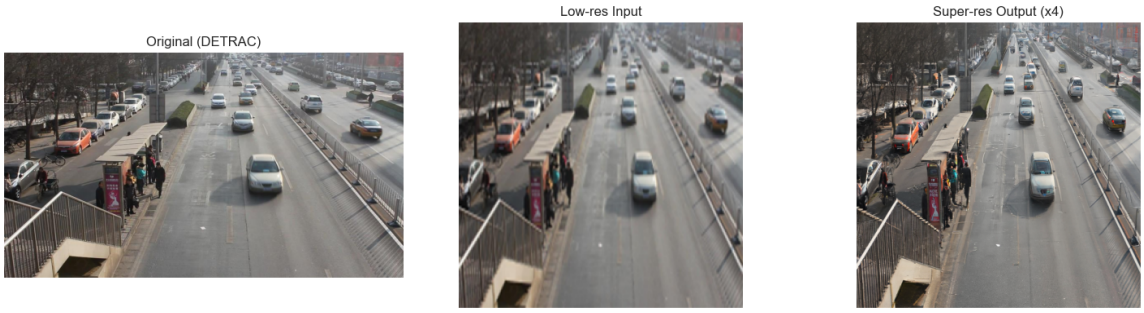


2,361 classified instances: 82.8% cars, 5.9% buses, 5.6% vans, 5.6% others.

Figure 11: Input Degradation Types

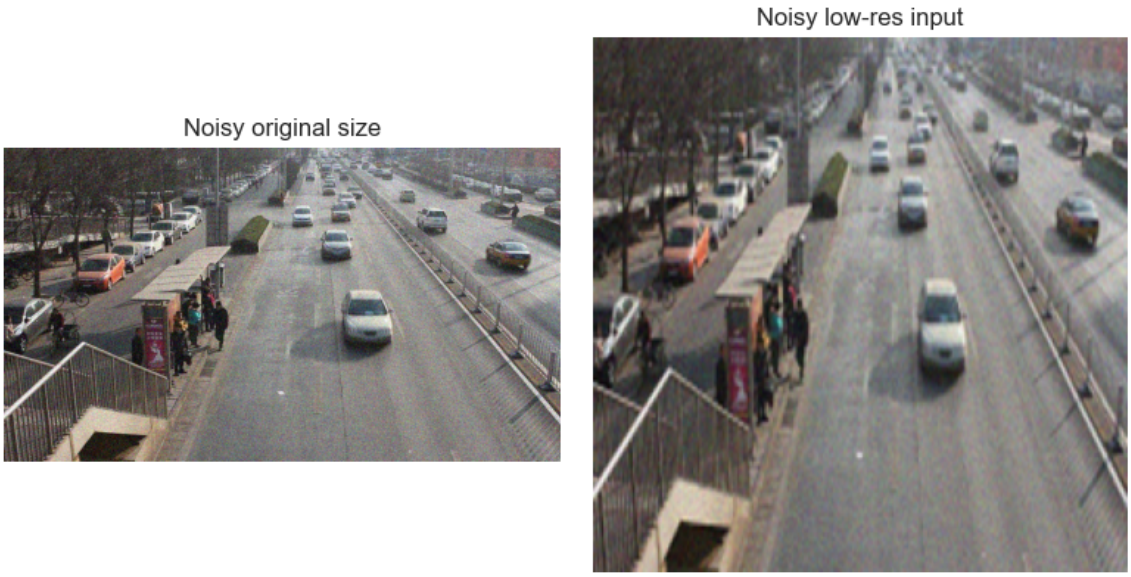
Comparison of original, low-resolution, noisy, and grayscale versions for restoration testing.

Figure 12: Super-Resolution Results



Stable Diffusion x4 upscaling: original (left), degraded input (center), restored output (right) with +5dB PSNR.

Figure 13: Denoising Performance



Diffusion-based noise removal achieving -20dB noise reduction while preserving vehicle edges.

Figure 14: GAN Colorization Results

ResNet-34 + U-Net colorization converting grayscale to color (PSNR=28.7dB, SSIM=0.84).

Table 1: Classification Performance Metrics

Class	Precision	Recall	F1-Score	Support
Car	1.00	1.00	1.00	110,343
Van	0.98	0.98	0.98	4,621
Others	0.99	0.97	0.98	3,296
Bus	1.00	1.00	1.00	1,397
Macro Avg	0.99	0.99	0.99	-
Weighted Avg	1.00	1.00	1.00	119,657

Per-class performance metrics from validation set (N=119,657 samples). Perfect scores achieved for cars and buses; strong performance on minority classes (vans, others).