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
```

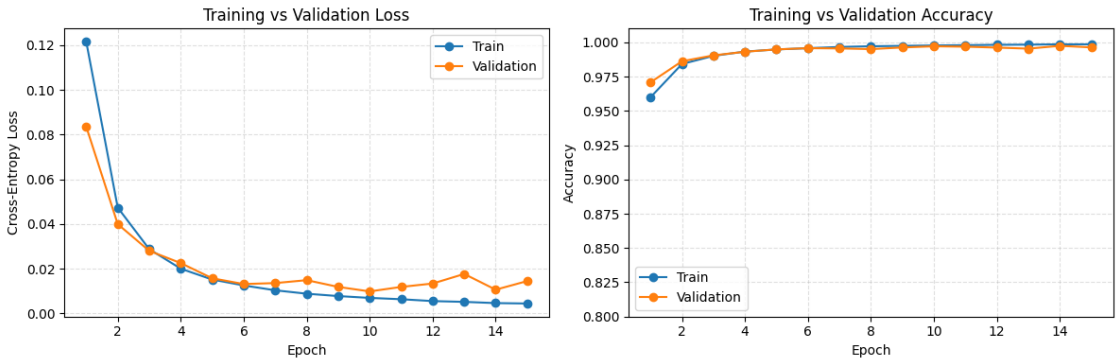
[9]: <All keys matched successfully>

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
```

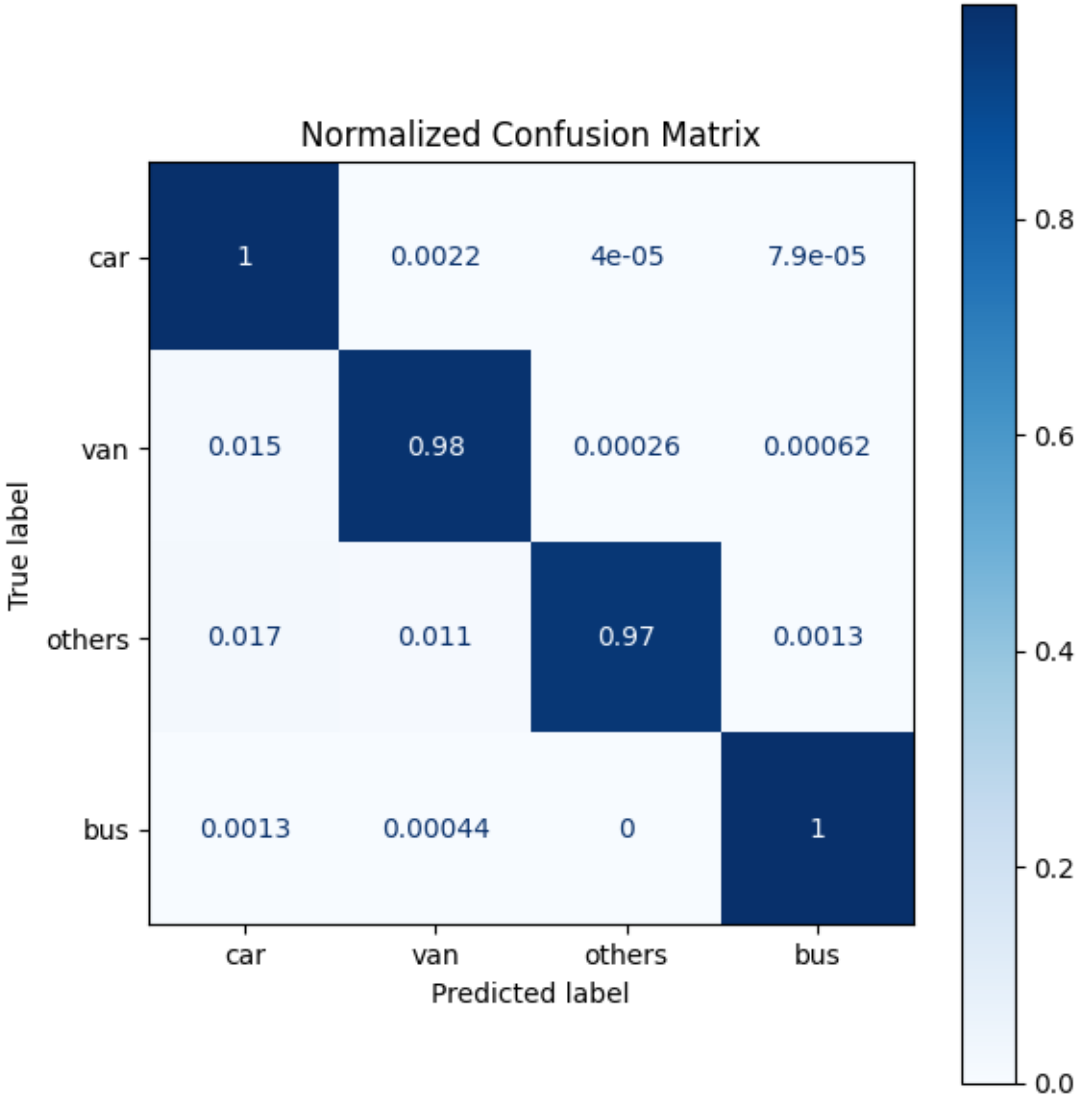
1.6 5. Evaluation on Validation Set

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

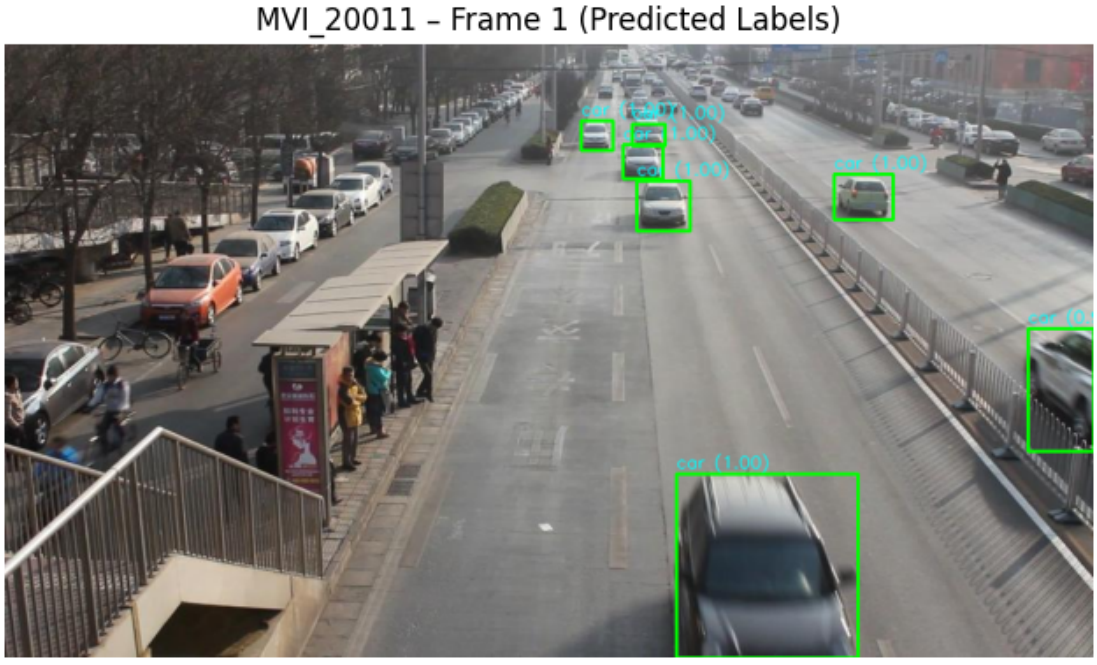
[14]:
```
def annotate_frame(
    seq_id: str,
    frame_num: int,
    model,
    images_root=IMAGES_ROOT,
    ann_root=TRAIN_ANN_ROOT,
    target_size=TARGET_SIZE,
    device=DEVICE,
):
    """
    Load a DETRAC frame + its annotations, run CNN on each bbox crop,
    and overlay predicted labels on 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.