Car Object Detection and Classification Final Report

Jannet Akanksha Ekka, Adari Pooja

1. Summary of problem statement, data and findings

Problem Statement: Design a deep learning-based car identification model that can detect and classify cars based on their make, model, and year.

Domain: Automotive Surveillance

The dataset's size posed challenges initially, hence, test chunks of the datasets were created to easily manage the data. Mappings of images with their classes and annotations were created for easy computational purposes. A sample dataset of 16 classes were utilized for experimentation.

Initially base CNN models like Basic CNN, VGG16 were designed, finetuned, trained and tested along with few ImageNet models namely MobileNetV2 and ResNet50. Additionally, EfficientNetV2B0 model was also experimented with to observe its take on this dataset.

ResNet50 model has outperformed others with 60% accuracy score just with the base model and was chosen for further fine tuning using Hyperparameter tuning technique which had an accuracy boost of ~70%. Predictions made by this model have shown promising results for this project task.

For Object Detection task, FastRCNN Lite models were experimented with ResNet50 and MobileNetV2 models as backbone. These models were designed, trained and evaluated on the given Stanford Cars dataset for both classification and regression. The Fast RCNN with MobileNetV2 model has given promising results among all the experiments and is proposed as our final solution for this task.

Sample Image
Class: Volkswagen Golf Hatchback 2012

2. Overview of the Final process

Dataset Overview:

- Cars dataset containing 16,185 images across 196 classes
- Split into 8,144 training images and 8,041 testing images
- Classes defined at Make, Model, Year level (e.g., "Volkswagen Golf Hatchback 2012")
- Each image includes annotations with bounding box coordinates and corresponding class label
- ~50-50 split per class between training and testing sets

Data Pre-Processing:

- Mappings between images, classes and annotations
- Dataset split into smaller, manageable chunks
- Data resizing and normalisation according to the models' needs

Basic CNN Models Experimented:

- Basic CNN
- VGG16
- ResNEt50
- MobileNetV2
- EfficientNEtV2B0

Fine-Tuning Approach on Best performing CNN models:

- HyperParameter Tuning with RandomSearch and Hyperband seach algorithms
- Data-Augmentation
- Modification Model layers
- Callbacks

RCNN Models Used:

- Fast RCNN using MobileNetV2 as backbone
- Fast RCNN using ResNet50 as backbone

Metrics and Predictions:

- Class level accuracies
- Model Architectures observed
- Train and Validation Accuracies compared
- Classification Report generated for best model

3. Step-by-step walk through the solution

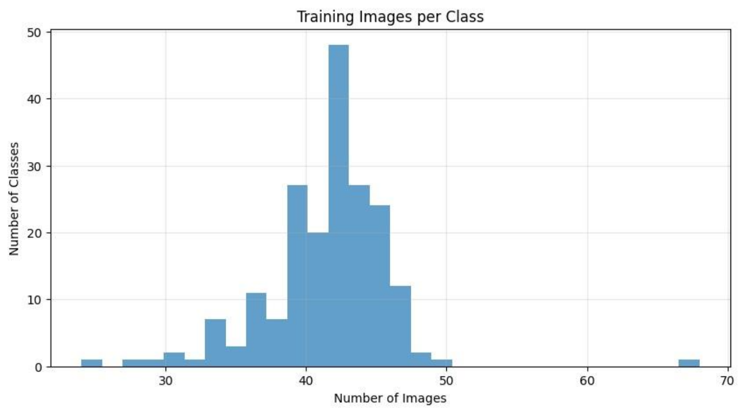
Data Organization:

Created structured mappings between:

- Images and their respective classes
- Images and their bounding box annotations
- Class names and numerical indices

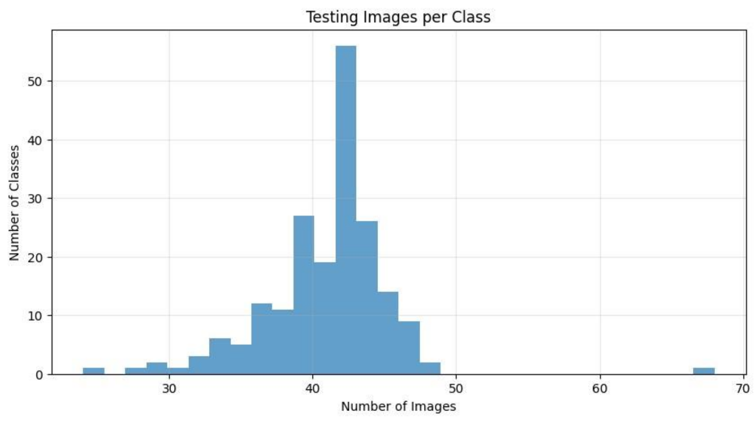
Data Distribution Analysis

Train dataset**:**



- Min images per class: 24
- Max images per class: 68
- Mean images per class: 41.6

Test dataset**:**



- Min images per class: 24
- Max images per class: 68
- Mean images per class: 41.0

Data Pre-processing Steps:

- Image Standardization:
    - Re-sized all images to 224x224 pixels
    - Normalized pixel values to range [0,1] ○

Maintained aspect ratio during resizing ● Data
Organization:

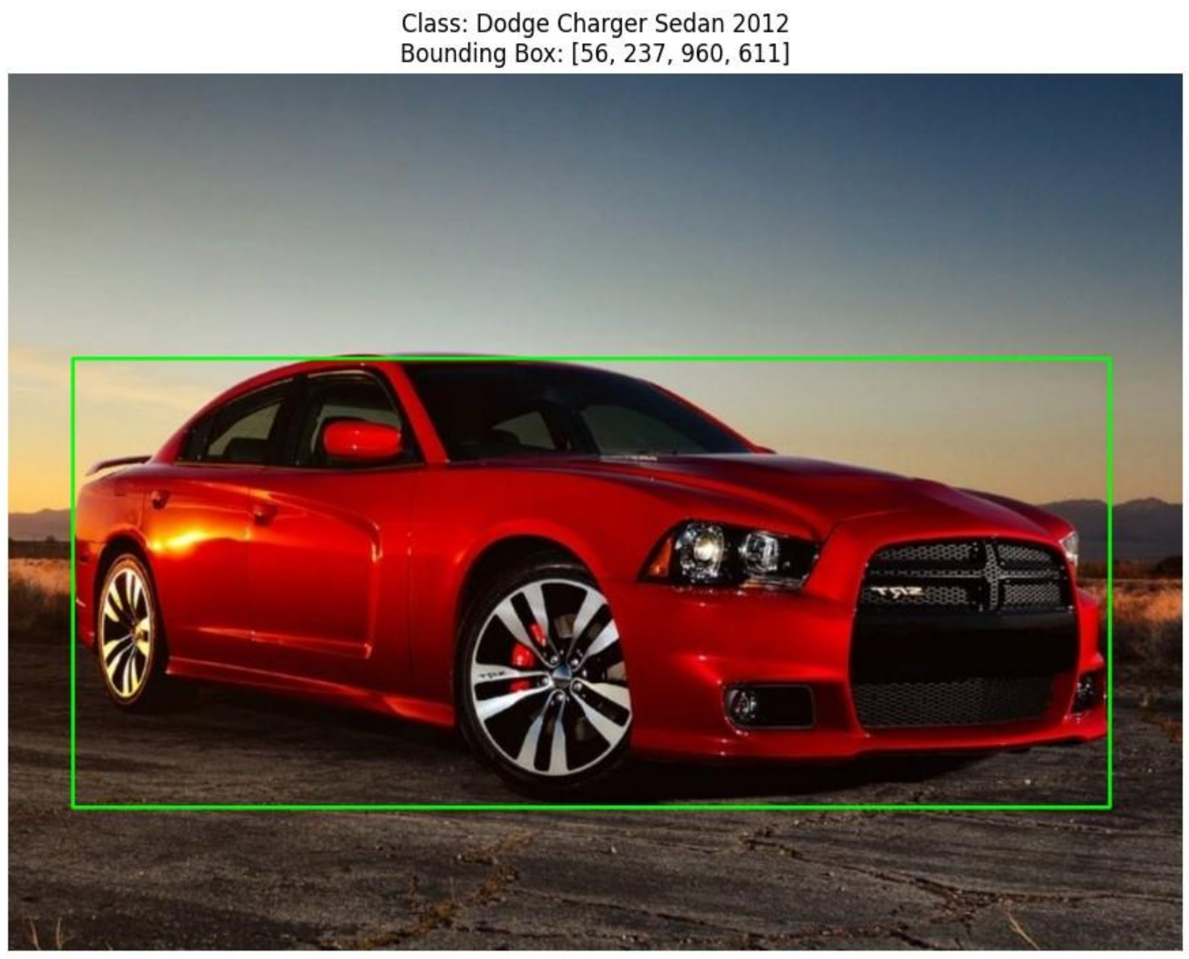
○ Implemented chunk-based processing to handle memory constraints
○ Created TensorFlow datasets for efficient training
○ Batch size: 32 images
○ Added shuffling for training data
○ Created mappings of the images with classes ●

Annotation Processing:

○ Verified bounding box coordinates
○ Ensured all coordinates are within image dimensions
○ Created mappings for easy access during training ●

Visualizations:

○ Displayed sample images with bounding boxes
○ Verified correct class labeling
○ Confirmed bounding box accuracy
○ Checked image quality and resolution distribution


Class: Dodge Charger Sedan 2012
Bounding Box: [56, 237, 960, 611]

Approach to Model Designing

For our car classification task, we adopted a two-pronged strategy to establish a comprehensive understanding of model performance:

1. Basic CNN Implementation
   - Develop a CNN model from scratch to understand fundamental deep learning concepts
   - Establish baseline performance metrics for comparison
   - Gain insights into dataset challenges and model behavior
   - Identify limitations when learning features without pre-training

2. Transfer Learning Implementation
   - Leverage pre-trained models that have proven success on ImageNet
   - Utilize established architectures of varying complexities
   - Take advantage of features learned from millions of images
   - Address limited dataset challenges through knowledge transfer

3. Fast RCNN Implementation
   - Use of transfer learning with pre-trained backbones
   - Two-stream network: Image processing and ROI processing
   - Integration of bounding box regression with classification
   - Choice of ResNet50 and MobileNetV2 as backbones
   - Batch size optimization for free resources ○ Data augmentation and preprocessing strategies

4. Resource-Constrained Implementation:

   - Working with data chunks due to computational limitations
   - Used 16 classes with ~29 samples per class
   - Total working dataset: 650 training and 642 testing samples
   - Classes selected based on sample availability and balance.

This dual approach enables us to:

- Compare effectiveness of learned vs pre-trained features
- Understand the impact of the model complexity on our specific dataset ● Evaluate the resources' requirements and their performance trade-offs
- Follow the best practices for car classification and regression tasks within computational constraints

The FINE-TUNING Approach:

For Fine-tuning, HyperParameter tuning approach is chosen. Keras Tuner framework from Tensorflow/Keras models is used for optimising the hyperparameters. It works best with Tensorflow models which have been used in this project.

Below are a few parameters considered for this task.

- Learning rate ○ Dropout rate
- Dense layer units

Extra layers such as below are also added to the models.

- Dense layers with extra units ○ Dropout
- BatchNormalisation
- Internal layers have the 'ReLu' activation added

Below search algorithms were incorporated for finding the best parameter combination of the hyperparameters. ○ Hyperband

- RandomSearch

The model training also had the below callbacks implemented for accuracy boost.

- ReduceLROnPlateau ○ Early stopping ○ ModelCheckpoint: To save the model with the improved val_accuracy for further runs.

RCNN Model Development

- Backbone architecture selection and freezing strategies
- Integration of ROI pooling layer
- Implementation of classification head
- Memory optimization techniques

RCNN Training Process

- Learning rate strategy with ReduceLROnPlateau
- Early stopping implementation
- Model checkpointing and saving

RCNN Evaluation Pipeline

- Implementation of FastRCNNEvaluator for Bounding box predictions
- Metrics calculation
- Visualization of the predictions
- Deciding the Best model based on the final metrics and predictions

YOLO V3 Pretrained Model Bounding Boxes Predictions

- YOLO correctly detects the Toyota Sequoia as a car with high confidence scores (0.90-0.95)
- The bounding boxes from YOLO (green) closely match the ground truth boxes (blue)
- The ground truth boxes (blue) are consistently well-placed, tightly bounding the vehicles
- They include all parts of the vehicle including wheels and roof racks
- They maintain consistent placement across different views of the same model

4. Model Evaluations

Below aspects were taken as part of the model evaluations done for the Fast RCNN models. These were considered as they would lead on to the best solution/model for this task.

Performance Metrics

- Accuracy scores for both architectures ○ Training vs validation performance
- Convergence analysis

Evaluation Methods

- Test set performance ○ Visualization of predictions ○ Error analysis
- Saving the model at its best val_accuracy

Model Comparison ○ ResNet50 vs MobileNetV2 performance ○ Training time comparison

- Prediction accuracy

5. Comparison to benchmark

Initial Basic CNN Model Evaluation results:

Model Version	Parameters	Training Time (min)	Best Train Acc (%)	Best Val Acc (%)	Epochs to Converge
Basic CNN	532,176	~25	26.50	7.59	14
VGG16	14,987,600	~60	98.74	51.62	20
ResNet50	24,647,056	~45	99.09	62.2	20
MobileNetV2	2,924,112	~27	99.82	59.17	20
EfficientNetv2B0	117,767,344	~45	6.29	6.97	10

Class Accuracies achieved with Basic CNN models

Model	Class Accuracy
Basic CNN	1: 0.00% (0/44 samples) 2: 0.00% (0/32 samples) 3: 0.00% (0/43 samples) 4: 0.00% (0/42 samples) 5: 0.00% (0/40 samples) 6: 0.00% (0/44 samples) 7: 0.00% (0/39 samples) 8: 0.00% (0/45 samples) 9: 17.07% (7/41 samples) 10: 0.00% (0/33 samples) 11: 0.00% (0/38 samples) 12: 0.00% (0/36 samples) 13: 0.00% (0/41 samples) 14: 100.00% (42/42 samples) 15: 0.00% (0/43 samples) 16: 0.00% (0/43 samples)
VGG16	1: 100.00% (44/44 samples) 2: 40.62% (13/32 samples) 3: 60.47% (26/43 samples) 4: 52.38% (22/42 samples) 5: 57.50% (23/40 samples) 6: 70.45% (31/44 samples) 7: 76.92% (30/39 samples) 8: 42.22% (19/45 samples)
	9: 46.34% (19/41 samples) 10: 39.39% (13/33 samples) 11: 81.58% (31/38 samples) 12: 75.00% (27/36 samples) 13: 75.61% (31/41 samples) 14: 40.48% (17/42 samples) 15: 69.77% (30/43 samples) 16: 81.40% (35/43 samples)
ResNet50	1: 100.00% (44/44 samples) 2: 43.75% (14/32 samples) 3: 55.81% (24/43 samples) 4: 59.52% (25/42 samples) 5: 55.00% (22/40 samples) 6: 79.55% (35/44 samples) 7: 69.23% (27/39 samples) 8: 71.11% (32/45 samples) 9: 58.54% (24/41 samples) 10: 51.52% (17/33 samples) 11: 86.84% (33/38 samples) 12: 66.67% (24/36 samples) 13: 68.29% (28/41 samples) 14: 59.52% (25/42 samples) 15: 76.74% (33/43 samples) 16: 93.02% (40/43 samples)

MobileNetV2	1: 100.00% (44/44 samples) 2: 43.75% (14/32 samples) 3: 62.79% (27/43 samples) 4: 59.52% (25/42 samples) 5: 52.50% (21/40 samples) 6: 65.91% (29/44 samples) 7: 64.10% (25/39 samples) 8: 68.89% (31/45 samples) 9: 41.46% (17/41 samples) 10: 33.33% (11/33 samples) 11: 84.21% (32/38 samples) 12: 75.00% (27/36 samples) 13: 75.61% (31/41 samples) 14: 42.86% (18/42 samples) 15: 74.42% (32/43 samples) 16: 86.05% (37/43 samples)
EfficientNetv2	1: 0.00% (0/44 samples) 2: 0.00% (0/32 samples) 3: 0.00% (0/43 samples) 4: 0.00% (0/42 samples) 5: 7.50% (3/40 samples) 6: 9.09% (4/44 samples) 7: 5.13% (2/39 samples) 8: 2.22% (1/45 samples) 9: 2.44% (1/41 samples) 10: 24.24% (8/33 samples) 11: 2.63% (1/38 samples) 12: 2.78% (1/36 samples) 13: 24.39% (10/41 samples) 14: 0.00% (0/42 samples) 15: 30.23% (13/43 samples) 16: 2.33% (1/43 samples)

Fine-tuned Models Evaluation Results

Model	Best Val Accuracy	Final Train Accuracy	Total Epochs
ResNet50_ft	0.6920	0.9923	20
MobileNetV2_ft	0.3467	0.9969	17
EfficientNetV2_ft	0.456	0.9843	20

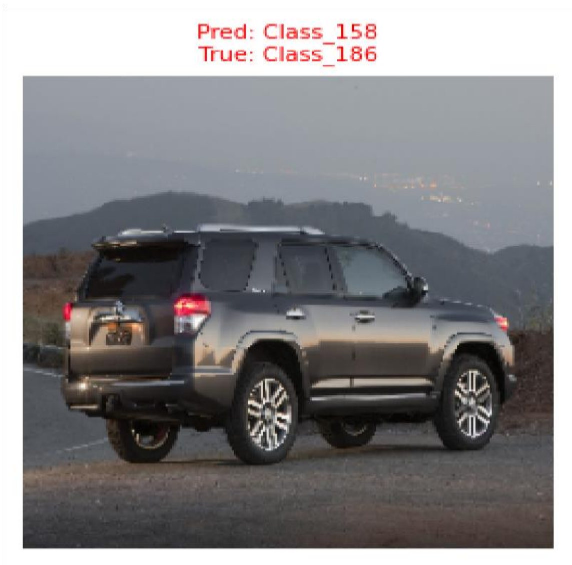
Fast RCNN Model Evaluation Results

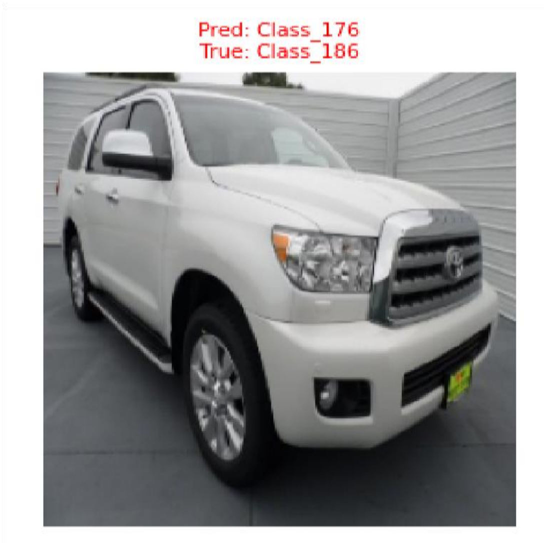
Model	Best Val Accuracy	Final Train Accuracy	Test Accuracy	Total Epochs
MobileNetV2_FastRCNN	0.5859	0.9406	0.5805	20
ResNet50_FastRCNN	0.2328	0.6172	0.2307	20

6. Visualisations

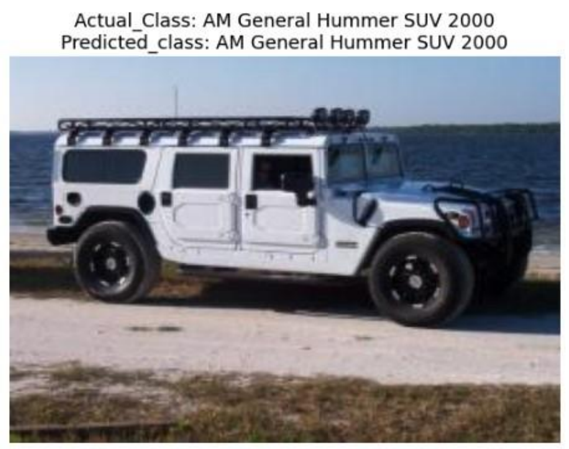
Basic CNN Predictions:

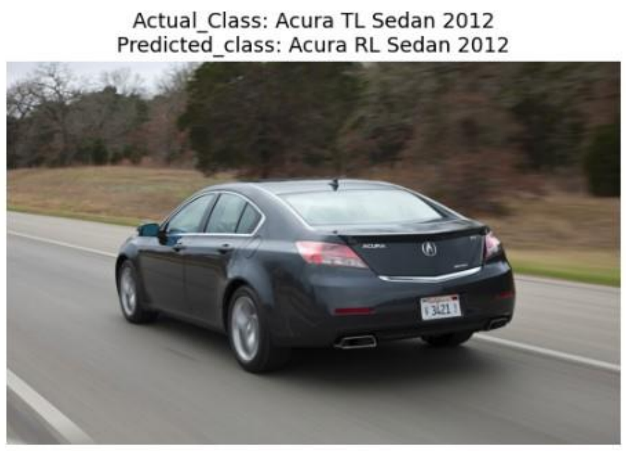
- Model shows bias towards certain classes (158, 176)
- Consistent misclassification patterns observed
- Poor generalization across most classes
- Better performance on visually distinctive vehicles



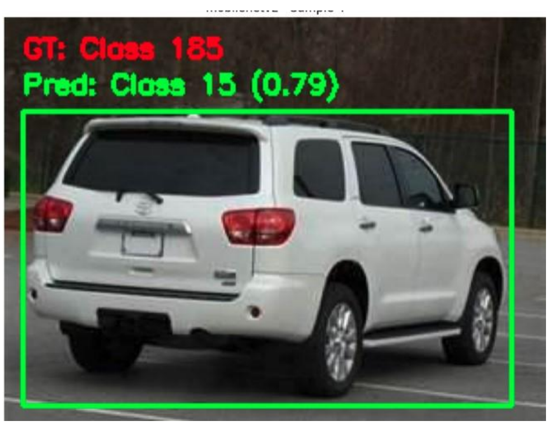


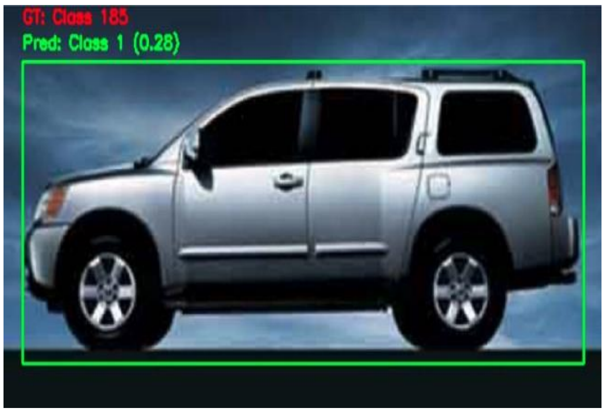
Fine Tuned ResNet50 Predictions:


Actual_Class: AM General Hummer SUV 2000
Predicted_class: AM General Hummer SUV 2000

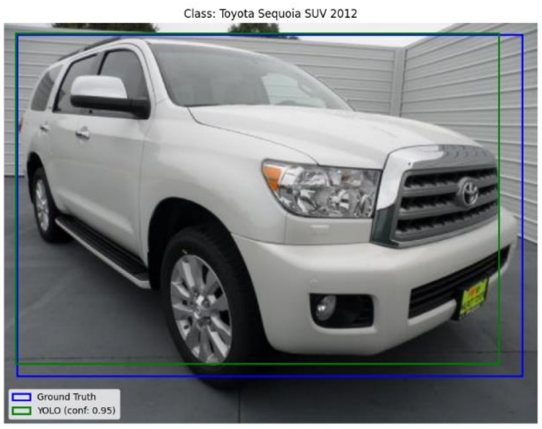

Actual_Class: Acura TL Sedan 2012
Predicted_class: Acura RL Sedan 2012

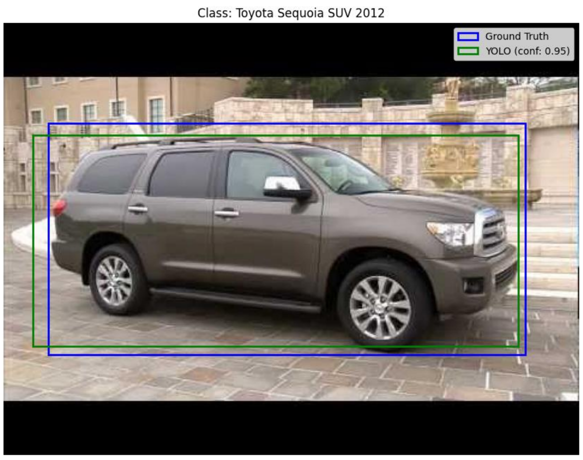
Fast RCNN with MobileNetV2 Backbone Predictions





Detecting bounding boxes with pretrained YOLO V3





7. Implications

Technical Implications

- Feasibility of Fast R-CNN for automotive surveillance
- Resource requirements and optimization

8. Limitations

Technical Limitations

- Memory constraints with free resources
- Processing speed considerations
- Backbone architecture limitations

Dataset Limitations

- Class imbalance issues
- Image quality variations
- Annotation accuracy impacts

9. Closing Reflections

Implementation of our Learnings

- Management of large datasets
- Batch processing
- Different approaches of model designing
- Effective code management for a better organised code
- Effectiveness of memory optimization strategies
- Impact of backbone choice

Future Improvements

- Potential for model optimization
- Additional backbone architectures to explore
- Data augmentation strategies

End Of Report