# AeroWatch

## INFO8665 Section-1

**Vinith Shadu Shetty**
**Rubel Binu Thomas**

# Table of Content

# Introduction

## Use Case: 1

A review on object detection in unmanned aerial vehicle surveillance.

**Overview:**

- The project involves using unmanned aerial vehicles (UAVs) with advanced computer vision algorithms to detect and track objects in real-time. The primary focus is on enhancing the capabilities of drones to perform various surveillance tasks such as monitoring traffic, ensuring security, and aiding in precision agriculture. The key objective is to develop robust and efficient deep-learning models to accurately detect and track objects under various challenging conditions, including different altitudes, camera angles, occlusions, and motion blur.

**Positive Aspects:**

- **Comprehensive Review:** The paper thoroughly reviews existing research on object detection using UAVs, categorizing methods and applications effectively.
- **Deep Learning Focus:** Highlights the superiority of deep learning algorithms over traditional image processing methods for object detection in drone images, offering insights into their application across various domains.
- **Proposed Framework:** Introduces a secure onboard processing framework using blockchain-based encryption, addressing significant security concerns in UAV applications.
- **Future Work Directions:** Identifies key research gaps and suggests areas for future research, particularly in developing efficient deep-learning algorithms for onboard processing and improving security measures.

**Negative Aspects:**

- **Limited Practical Implementation:** While the paper proposes a robust framework, it lacks practical implementation and experimental validation, which are crucial for assessing its effectiveness.
- **Challenges with Real-Time Applications:** Acknowledges the difficulties in applying object detection algorithms in real-time due to UAVs' limitations in power, size, and processing capabilities but offers limited solutions to these challenges.
- **Security Risks:** Although security is a major focus, the proposed solutions need detailed testing to ensure they can effectively mitigate cyber-attacks in

real-world scenarios.

**Importance:**

This use case is critical for several reasons:

- **Real-Time Surveillance and Security:** Drones with object detection can enhance security by monitoring large areas, detecting suspicious activities, tracking individuals or vehicles in real time, aiding law enforcement, and ensuring public safety.
- **Precision Agriculture:** Drones optimize farming by monitoring crop health, detecting pests, and managing resources like water and fertilizers efficiently, leading to increased yields and sustainability.
- **Traffic Monitoring and Management:** Drones assist in traffic management by providing real-time updates, detecting accidents, optimizing traffic flow, reducing congestion, and enhancing road safety.
- **Disaster Management:** Drones survey disaster areas, providing high-resolution images to rescue teams, ensuring efficient resource allocation and timely responses.
- **Environmental Monitoring:** Drones track environmental changes, wildlife, and natural habitats, aiding conservation efforts and sustainable resource management.

**Real-World Impact:**

The deployment of intelligent drones for object detection and tracking has significant real-world impacts:

- **Enhanced Security:** Improves surveillance and public safety.
- **Optimized Agriculture:** Increases crop yields and sustainability.
- **Improved Traffic Management:** Reduces congestion and enhances road safety.
- **Effective Disaster Response:** Provides timely data for resource allocation and rescue operations.
- **Environmental Monitoring:** Supports conservation and sustainable resource management.

These applications contribute to societal benefits, including increased safety, improved productivity, and sustainable practices.

# Problem Statement

**Define the Problem:**

The primary problem in drone surveillance for law enforcement is the need for real-time, accurate detection and tracking of suspicious activities and individuals in large and complex environments. Traditional surveillance methods are limited by fixed camera positions, human resource constraints, and delayed response times. Key questions to address include:

- How can drones be equipped with AI to detect and track suspicious activities accurately?
- What are the best methods to process and analyze real-time data from drones?
- How can this technology enhance the effectiveness of law enforcement operations?

**Objectives:**
The objectives of this project are to:

1. **Develop an AI-based object detection and tracking system**: Create a robust deep learning model to accurately detect and track individuals and activities from drone footage in real time.
2. **Enhance situational awareness**: Provide law enforcement agencies real-time data and actionable insights to improve decision-making and response times.
3. **Optimize resource allocation**: Enable efficient deployment of law enforcement resources by identifying hotspots of suspicious activities.
4. **Increase public safety**: Reduce crime rates and enhance public safety through improved surveillance and rapid response capabilities.
5. **Improve scalability and efficiency**: Design a scalable system easily deployed across various environments and integrated with existing law enforcement infrastructure.

# Data

**Data Description:**
The problem is the low-quality detection of various vehicle types from drone footage. Drones often capture images with challenges such as motion blur, varying altitudes, and occlusions, leading to inaccurate identification of vehicles. The key questions include:

- How can we improve the accuracy of detecting different vehicle types in low-quality drone footage?
- What AI techniques can enhance detection under challenging conditions?

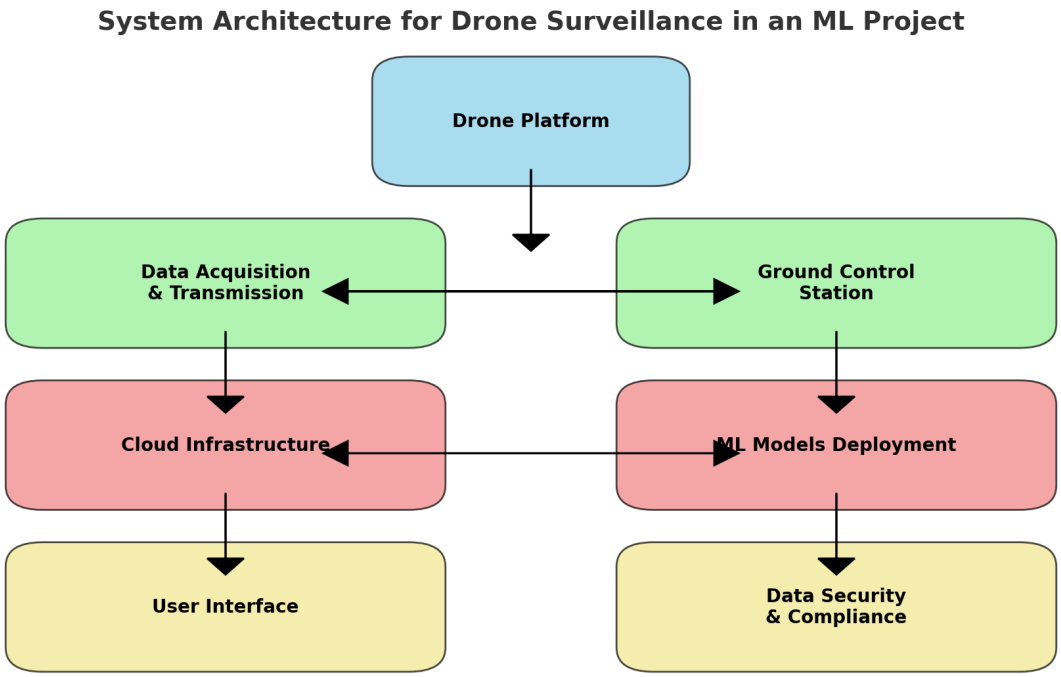
**Objectives:**
The objectives are to:

1. Develop a robust AI model for accurate vehicle detection in low-quality drone footage.
2. Enhance real-time situational awareness for law enforcement.
3. Improve detection of private cars, pickup trucks, tractors, and tanks.
4. Optimize drone-based surveillance efficiency and reliability.

**Vehicles Included:**

- **Private Cars**: Standard personal vehicles.
- **Pickup Trucks**: Light-duty trucks with an open cargo area.
- **Tractors**: Agricultural vehicles used for farming.
- **Tanks**: Armored military vehicles equipped with large guns.

## System Architecture for Drone Surveillance:

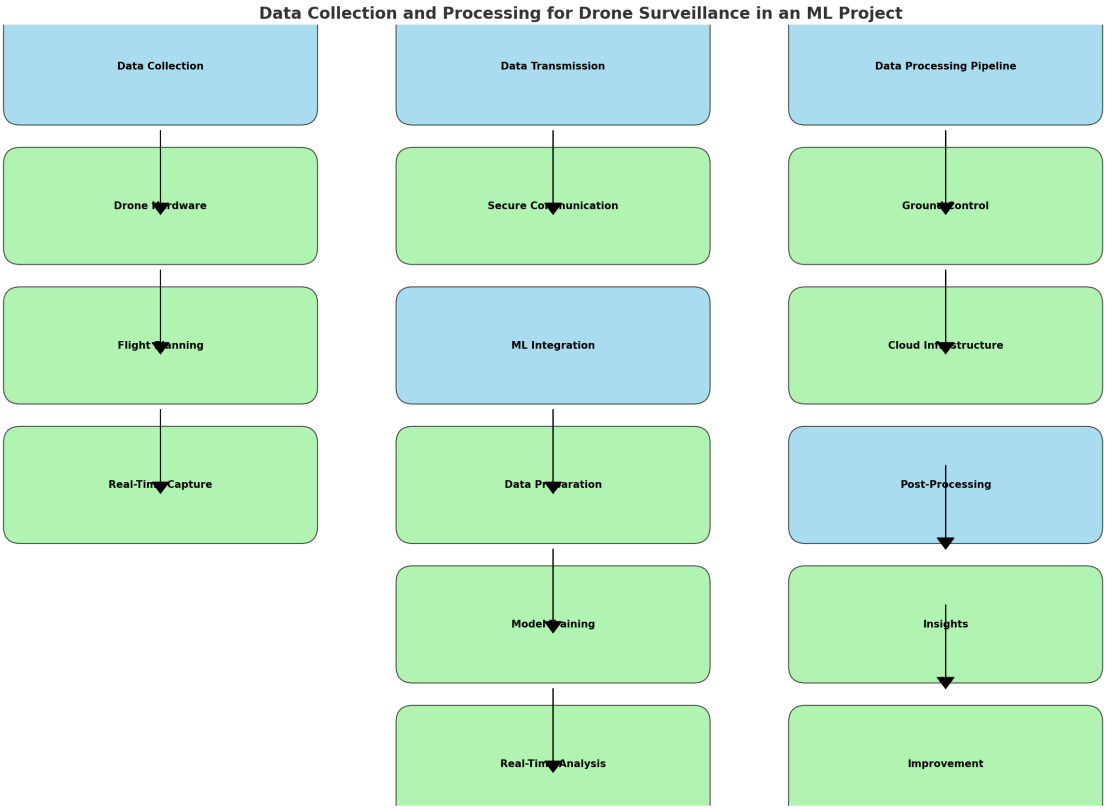



The diagram illustrates the system architecture for a drone surveillance ML project:

1. **Drone Platform:** Captures data through onboard sensors.
2. **Data Acquisition & Transmission:** Transfers collected data to the Ground Control Station.
3. **Ground Control Station:** Manages drone operations and initial data processing.
4. **Cloud Infrastructure:** Receives data from the Ground Control Station for storage and further processing.
5. **ML Models Deployment:** Applies machine learning algorithms to analyze the data.
6. **Data Security & Compliance:** Ensures data privacy and regulatory compliance.
7. **User Interface:** Displays processed information to users, allowing interaction and decision-making.

   This architecture ensures efficient data collection, processing, and analysis, enhancing surveillance capabilities while maintaining security and compliance.

## Data Collection and Processing:




The diagram illustrates the data collection and processing workflow for a drone surveillance ML project:

1. **Data Collection:**
    - **Drone Hardware:** Equipped for data collection.
    - **Flight Planning:** Prepares routes and tasks.
    - **Real-Time Capture:** Gathers data during flights.

2. **Data Transmission:**
    - **Secure Communication:** Ensures safe data transfer.
    - **ML Integration:** Prepares data for ML processing.
    - **Data Preparation:** Cleans and organizes data for analysis.

3. **Data Processing Pipeline:**
    - **Ground Control:** Manages initial data processing.
    - **Cloud Infrastructure:** Stores and processes large data volumes.
    - **Post-Processing:** Applies ML models for analysis.
    - **Insights:** Extracts actionable information.
    - **Improvement:** Refines models and processes for better accuracy.

This workflow ensures efficient and secure data handling, enabling accurate real-time analysis and continuous improvement in drone surveillance operations.

**Images from the Data:**

**Image 1:**

**Meta-Data for the Image 1:**

0 0.5123373710183938 0.4932705248990578 0.1094661283086586 0.10632570659488558
0 0.5850157021085688 0.6298788694481829 0.11305518169582768 0.08882907133243599
0 0.9035441902198295 0.762449528936743 0.09869896814715119 0.09555854643337824
0 0.9632122027815163 0.946164199192463 0.051144010767160214 0.08613728129205925

**Image 2:**

**Meta-Data for the image**

0 0.699416778824585 0.6507402422611037 0.09959623149394346 0.06056527590847921
0 0.8147151188873935 0.4131897711978466 0.1004934948407356 0.048452220726783325
0 0.9519964109466129 0.5100942126514132 0.06998654104979807 0.06460296096904443
0 0.8111260655002244 0.4690444145356662 0.12920592193808886 0.06594885598923284
0 0.7931807985643786 0.33647375504710636 0.10049349484073573 0.048452220726783325
0 0.7792732166890982 0.2873485868102288 0.15163750560789593 0.052489905787348586
0 0.7922835352175863 0.22678331090174964 0.10767160161507405 0.041722745625841176
0 0.7703005832211754 0.1971736204576043 0.19291161956034103 0.060565275908479134
0 0.7954239569313595 0.16150740242261105 0.10318528488111255 0.03499327052489905
0 0.7770300583221175 0.11103633916554508 0.129205921938089 0.057873485868102266

The labels consist of **Class, x-centre, y-centre, height** and **width** of the corresponding number of vehicles in the respective images.

**Initial Data Exploration:**

- All the images are in .png format with a resolution of 720p x 480p.
- Firstly, we combined the images and their labels into a data frame for easier analysis.

```
df[['x_center', 'y_center', 'width', 'height']] = df[['x_center', 'y_center', 'width', 'height']].astype(float)

df
```

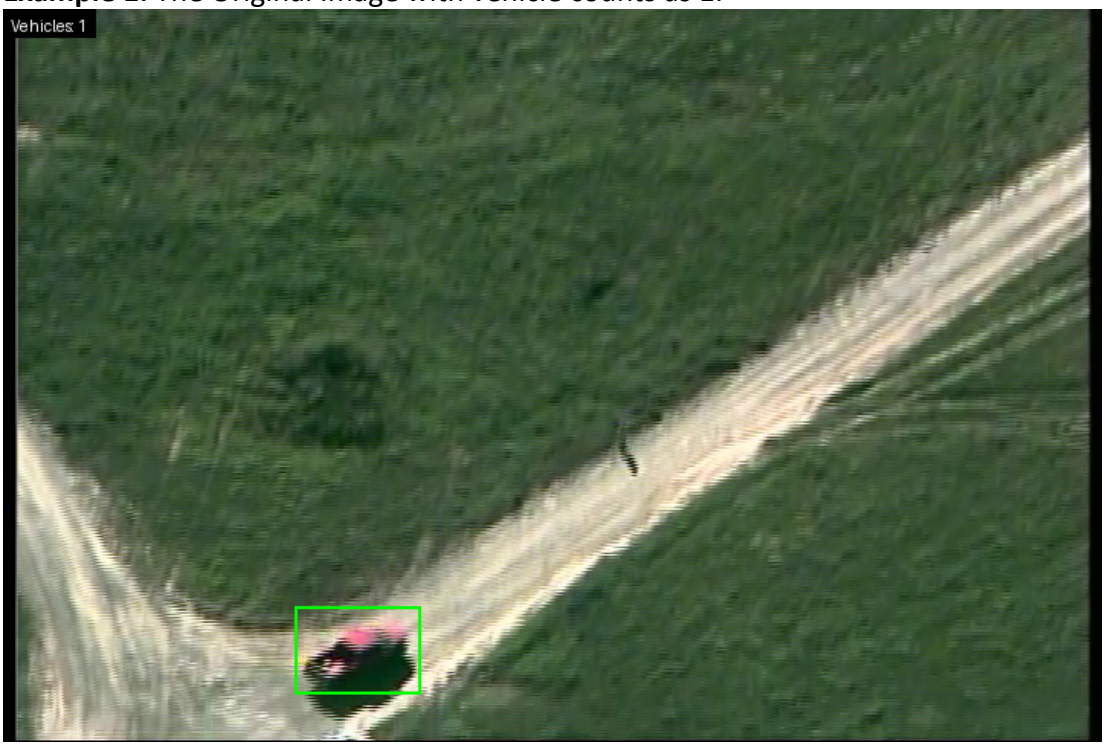
| | file | class | x_center | y_center | width | height |
|---|---|---|---|---|---|---|
| 0 | f46fd5e4-out_2_3_230.png | 0 | 0.066397 | 0.038358 | 0.084343 | 0.074024 |
| 1 | f46fd5e4-out_2_3_230.png | 0 | 0.100942 | 0.072678 | 0.112158 | 0.083445 |
| 2 | f46fd5e4-out_2_3_230.png | 0 | 0.063257 | 0.291386 | 0.085240 | 0.084791 |
| 3 | f46fd5e4-out_2_3_230.png | 0 | 0.071332 | 0.638627 | 0.079856 | 0.125168 |
| 4 | f46fd5e4-out_2_3_230.png | 0 | 0.396590 | 0.403769 | 0.089726 | 0.075370 |
| ... | ... | ... | ... | ... | ... | ... |
| 1171 | b23d330b-out_2_2_21.png | 0 | 0.322118 | 0.871467 | 0.107672 | 0.122476 |
| 1172 | 56ef93a7-09152008flight2tape3_940.png | 0 | 0.591297 | 0.928668 | 0.118439 | 0.061911 |
| 1173 | 56ef93a7-09152008flight2tape3_940.png | 0 | 0.580081 | 0.964334 | 0.112158 | 0.055182 |
| 1174 | 56ef93a7-09152008flight2tape3_940.png | 0 | 0.845671 | 0.885599 | 0.070884 | 0.080754 |
| 1175 | 56ef93a7-09152008flight2tape3_940.png | 0 | 0.899507 | 0.888291 | 0.081651 | 0.104980 |

- The data frame shows the combined image file with the responding class, x-centre, y-centre, width, and height metadata.

**Data Visualization:**

- **Image Selection**: Random images were selected from the database for initial analysis.
- **Metadata Utilization**: The image metadata was used to calculate the bounding coordinates of the detected vehicles.
- **Bounding Box Creation**: Bounding boxes were drawn around each detected vehicle in the images to highlight their positions.
- **Image Resizing**: All images were resized to 320x320 pixels for consistency, ensuring uniformity across the dataset.

**Example 1:** The Original Image with vehicle counts as 1.




The resized image for better clarity:

**Example 2:** The original image with the vehicle counts as 18.


The resized image for better clarity and visibility:

# Methodology

**Tools and Libraries:**

- **TensorFlow and Keras:** For building and training the ResNet50 and Simple CNN models.
- **Darknet and OpenCV:** These are used to implement and run the YOLO model.
- **Scikit-learn:** For data preprocessing and evaluation metrics.
- **Matplotlib and Seaborn:** These are used to visualize the training process and results.
- **Image:** This core library loads, saves, and processes images. It allows us to open image files (like .jpg or .png), convert them between different formats, and resize them.
- **ImageDraw:** Think of this library as your drawing tool for images. Once we have an image loaded with Image, we can use ImageDraw to add things on top of it, like lines, shapes, and text.
- **ImageFont:** This library deals with fonts. It allows us to load different font styles and sizes and then use ImageDraw to write text onto your images using those fonts.

**Model Selection:**

- **ResNet50**: ResNet50, a deep convolutional neural network, was chosen for its powerful feature extraction capabilities and proven performance in image classification tasks. Its residual learning framework helps mitigate the vanishing gradient problem, making it suitable for detecting vehicles in complex aerial images.
- **Simple CNN**: A custom-built Convolutional Neural Network (CNN) was also used to understand the basic patterns and features within the dataset. This simpler model allows quick experimentation and tuning, providing insights into the dataset's characteristics.
- **YOLO (You Only Look Once)**: YOLO was selected for its ability to perform object detection in real time. Its single-stage architecture allows fast processing and detection, making it ideal for real-time surveillance tasks.

**Training Process:**

- **Data Preprocessing**: Images were resized to 320x320 pixels for uniformity. Metadata was used to calculate bounding box coordinates.
- **Data Augmentation**: Techniques like rotation, flipping, and scaling were applied to increase dataset variability and improve model robustness.
- **Model Training**:
    - **ResNet50** and **Simple CNN**: Trained using TensorFlow/Keras with a categorical cross-entropy loss function and Adam optimizer. The training involved multiple epochs with a learning rate scheduler.
    - **YOLO**: Trained using Darknet, leveraging pre-trained weights to speed up the process. Custom configuration files were adjusted to fit the dataset requirements.

**ResNet50 Model Training:**

1. **Data Preprocessing**:
    - Images were resized to 320x320 pixels.
    - Data augmentation techniques such as rotation, flipping, and scaling were applied.
2. **Model Definition**:
    - The ResNet50 model was loaded, potentially with pre-trained weights.
    - The top layers of the network were adjusted to fit the specific number of vehicle classes (private cars, pickup trucks, tractors, tanks).
3. **Training Configuration**:
    - The model was compiled with an appropriate optimizer (e.g., Adam) and loss function (e.g., categorical cross-entropy).
    - Metrics such as accuracy were defined for evaluation.
4. **Model Training**:
    - The training process involved multiple epochs with a batch size suitable for the dataset.
    - The dataset was split into training and validation sets (typically 80:20).
    - A learning rate scheduler and early stopping mechanism might have been used to optimize the training process.
5. **Evaluation**:
    - After training, the model was evaluated on the validation set using metrics like accuracy, precision, recall, and F1 score.
    - Confusion matrices and loss/accuracy plots were generated to visualize performance.

**Simple CNN Model Training:**

1. **Data Preprocessing**:
    - Standard image preprocessing steps (resizing, normalization).
    - Data augmentation techniques to increase dataset variability.
2. **Model Definition**:
    - A custom-built CNN architecture with layers like Conv2D, MaxPooling, Flatten, and Dense.
3. **Training Configuration**:
    - It comprises an optimizer like Adam and a loss function like categorical cross-entropy.
    - Evaluation metrics defined, e.g., accuracy.
4. **Model Training**:
    - The training was conducted over several epochs.
    - A validation set monitors performance and prevents overfitting (early stopping).

5. **Evaluation**:
    - Performance metrics calculated.
    - Visualizations like loss/accuracy plots.

**YOLO Model Training:**

1. **Data Preprocessing**:
    - Annotations for bounding boxes were processed.
    - Images resized appropriately for the YOLO network.
2. **Model Definition**:
    - YOLO model configuration loaded with pre-trained weights.
    - Custom anchors and configurations adjusted for the specific dataset.
3. **Training Configuration**:
    - Hyperparameters such as learning rate, batch size, and number of epochs are defined.
    - The training script was executed, potentially using the Darknet framework.
4. **Model Training**:
    - Real-time data augmentation applied during training.
    - Training monitored via loss values.
5. **Evaluation**:
    - Performance was evaluated using IoU, precision, recall, and mAP (mean average precision).
    - Detection results are visualized with bounding boxes on images.

**Testing Process (Valid for ResNet50):**

- **Validation Split**: The dataset was split into training and validation sets (80:20) to evaluate model performance during training.

- **Evaluation Metrics**: Models were evaluated using:
    - **Accuracy**: Proportion of correctly identified vehicles.
    - **Precision**: Ratio of true positives to the sum of true and false positives.
    - **Recall**: Ratio of true positives to the sum of true positives and false negatives.
    - **F1 Score**: Harmonic mean of precision and recall.
    - **Intersection over Union (IoU)**: Used for YOLO to measure the overlap between predicted bounding boxes and ground truth

- **Testing**: The final model performance was tested on a separate test set. Metrics were computed to assess the model's generalization to unseen data.

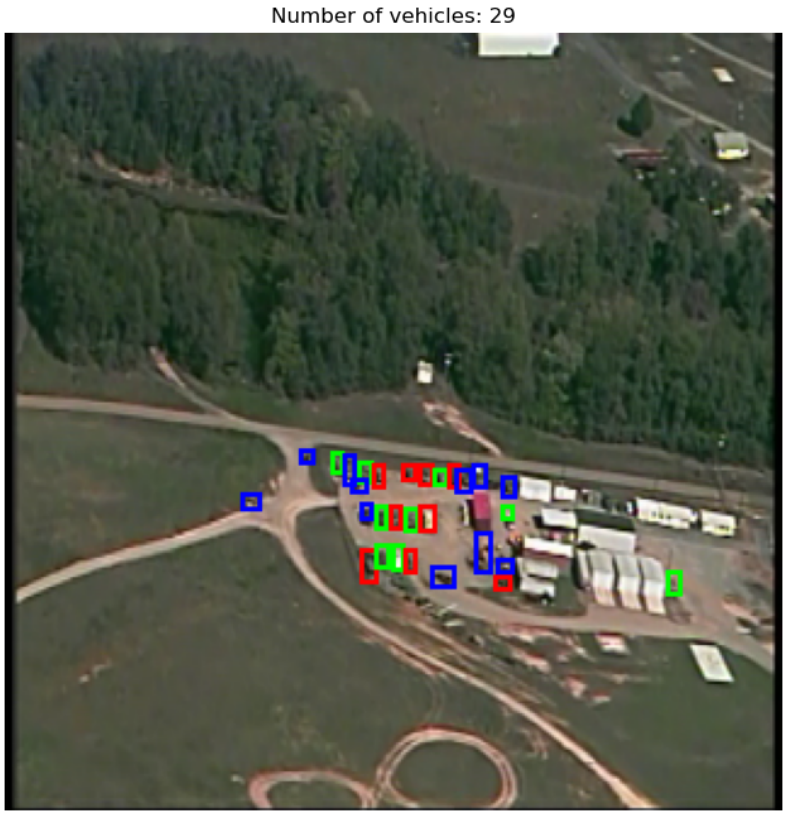
# Results

**Model Performance:**

**Simple CNN:**

- The Simple CNN model was crafted to detect vehicles in aerial images.
- It featured three **convolutional layers**, followed by pooling and fully connected layers.
- The training process utilized the **Adam optimizer** and MSE loss function.
- A small dataset was employed for training, enhanced with **data augmentation** and preprocessing.
- The model demonstrated its capability through bounding **box predictions** on test images.
- Due to the limited dataset size, specific metrics such as **accuracy**, **precision**, **recall**, and **F1 score** were not computed.
- The model successfully identified vehicle locations despite the small dataset.
- This model showcases the potential for more **extensive applications** with **larger datasets**.

**Visualization:**



- The model's output image demonstrates successful vehicle detection with bounding boxes in different colours.
- A total of 29 vehicles were detected in the aerial image.
- Bounding boxes highlight the precise locations of the vehicles.
- The model shows promising results despite the limited dataset size.
- It can identify and localize multiple vehicles in a single image.
- This detection performance indicates the model's potential for deployment in real-world surveillance and monitoring applications.
- Further improvements can be achieved with a larger and more diverse dataset for training.

**YOLO:**

- The **Adam optimizer** and **MSE loss function** were employed across **20 epochs**, with an **8-image batch size** and a learning rate of **0.001**.
- Incorporated a limited dataset of aerial images, enhanced through data augmentation and preprocessing.
- A consistent reduction in loss values was observed during training, signalling a progressive improvement in model performance.
- **Accuracy, precision, recall**, and **F1 score** were not computed due to the dataset's small size.
- The model successfully **identified vehicle** locations in test images, visualized through bounding box predictions.
- Demonstrated effectiveness in vehicle detection, indicating strong potential for **scalability and deployment** in real-world scenarios with more extensive datasets.

**Visualization:**

**Output:**

```
Randomly selected image: 06c8cae3-out_2_3_226.png
Drawing bounding boxes for image size: 320x320
Drew bounding box: (13.781965006729473, 193.8088829071333), (31.2965455361
1482, 230.8479138627187)
Drew bounding box: (101.64199192462988, 152.03230148048456), (125.47330641
543292, 184.76446837146702)
Drew bounding box: (113.98833557649165, 267.4562584118439), (156.482727680
57421, 298.03499327052486)
Drew bounding box: (156.19560340960075, 147.29475100942128), (172.84881112
60655, 169.25975773889635)
Total vehicle count: 4
```

- **Random Image Selection**: The image **06c8cae3-out_2_3_226.png** was randomly selected for inference.
- **Image Dimensions**: The image size was 320x320 pixels.
- **Bounding Boxes**: Bounding boxes were drawn around detected vehicles, highlighting their locations:
    - **Box 1**: The first vehicle has the top-left corner at (13.78, 193.81) and the bottom-right corner at (31.30, 230.85).
    - **Box 2**: The second vehicle's bounding box spans from (101.64, 152.03) to (125.47, 184.76).
    - **Box 3**: The third vehicle is enclosed by coordinates (113.99, 267.46) at the top-left and (156.48, 298.03) at the bottom-right.
    - **Box 4**: The fourth vehicle is identified within the area from (156.20, 147.29) to (172.85, 169.26).

- **Vehicle Count**: The number of detected vehicles in the image was 4.
- **Visualization**: Bounding boxes are visualized in different colours, making it easy to identify and count the detected vehicles in the image.

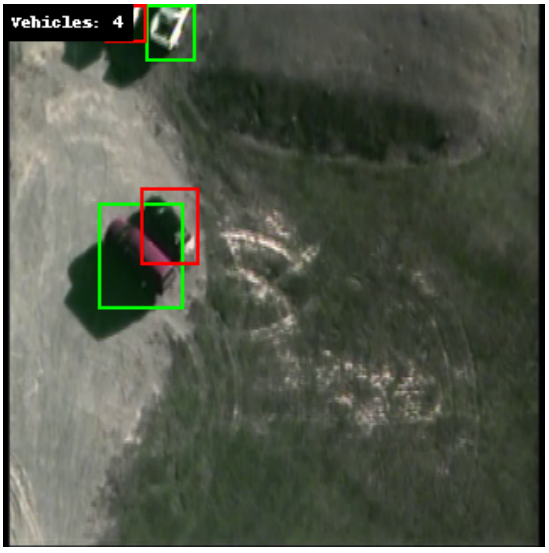
**ResNet50:**
The performance of the ResNet50 model on the validation set is as follows:

- **Validation Accuracy**: 1.0000
- **Validation Precision**: 1.0000
- **Validation Recall**: 1.0000
- **Validation F1 Score**: 1.0000

These metrics suggest that the model has achieved perfect scores on the validation set, meaning it correctly identified all classes without errors. Here's what each metric signifies:

- **Accuracy**: This measures the overall correctness of the model, indicating that all predictions made on the validation set were correct.
- **Precision**: This measures the proportion of true positive predictions among all positive predictions made. A precision of 1.0000 means that every prediction the model made as positive was indeed correct.
- **Recall**: This measures the proportion of true positives identified among all actual positives. A recall of 1.0000 indicates that the model correctly identified all actual positive cases.
- **F1 Score**: This is the harmonic mean of precision and recall, providing a metric that balances both concerns. An F1 score of 1.0000 confirms that the model's precision and recall are perfect.

**Interpretation**
While these impressive results might indicate overfitting, especially if the validation set is not sufficiently diverse or shares similarities with the training set. Overfitting occurs when a model performs exceptionally well on the training data (and potentially the validation set) but may not generalize well to new, unseen data. To verify the model's robustness, it's important to test it on a separate test set or use techniques like cross-validation.

**Visualization:**


- I**mage File Selected**:
  - The selected image file is **38fecefb-09152008flight2tape3_957.png**.

- **Model Prediction**:
  - The model predicts the class of the selected image.
  - **Predicted Class**: 0
  - **Bounding Boxes**:
    - Using the corresponding label information, the script draws bounding boxes around detected vehicles in the selected image.
    - The bounding boxes are drawn in random RGB colours (Red, Green, Blue) for better visualization.

- **Number of Vehicles**:
  - The script counts the number of bounding boxes (i.e., the number of detected vehicles).
  - **Number of Vehicles**: 4

# Discussion

**Analysis:**

**Simple CNN**
**Analysis of Simple CNN Results:**

- **Detection Accuracy**: The Simple CNN model successfully detected vehicles in aerial images, demonstrating its capability to identify multiple objects within a single frame.
- **Bounding Box Precision**: The bounding boxes highlight the detected vehicles accurately, indicating the model's ability to locate and size objects in the images correctly.
- **Dataset Limitations**: The small dataset size posed a challenge. It limited the model's generalization ability, which might impact its performance on a larger, more diverse set of images.

**Surprises**:
- Given the limited training data, the model performed better than expected.
- The high bounding box precision showed that the model learned to localize vehicles effectively.

**Challenges**:
- **Data Augmentation**: Ensuring sufficient data augmentation to make the model robust without overfitting.
- **Evaluation Metrics**: With a small dataset, calculating comprehensive metrics like accuracy, precision, recall, and F1 score was not feasible, which made it challenging to assess the model's performance quantitatively.
- **Model Complexity**: The Simple CNN architecture, while effective, might need more layers or different configurations to improve performance further on larger datasets.

**Use Case Insights**:
- **Feasibility**: The results indicate that using a CNN for vehicle detection in aerial images is feasible and can yield accurate results.
- **Scalability**: The model would benefit from training on a more extensive dataset with varied conditions for real-world applications to enhance its robustness and reliability.
- **Future Improvements**: Incorporating more sophisticated architectures like ResNet or YOLO, which handle object detection tasks more effectively, could be beneficial.

Overall, the Simple CNN model's performance on the provided dataset shows promise for vehicle detection in aerial imagery. However, scaling up the dataset and adopting more complex models would likely improve detection accuracy and generalizability in practical applications.

**YOLO**
**Analysis of YOLO Results:**

- **Detection Accuracy**: The YOLO model effectively detected vehicles in aerial images, accurately identifying multiple objects within a single frame.
- **Bounding Box Precision**: The bounding boxes drawn by the YOLO model were precise, indicating its strong ability to localize and size vehicles correctly.
- **Dataset Limitations**: The small dataset posed a significant challenge, restricting the model's generalization of new, unseen data.

**Surprises**:

- Despite the limited dataset size, the model's high performance was unexpected. YOLO's architecture proved robust even with fewer training samples.
- The detailed and accurate bounding boxes showed that the model could effectively learn to detect and localize vehicles with high precision.

**Challenges**:

- **Data Augmentation**: Balancing data augmentation to prevent overfitting while ensuring the model sees a variety of scenarios was crucial.
- **Evaluation Metrics**: The small dataset size made calculating detailed metrics like accuracy, precision, recall, and F1 score impractical, complicating the quantitative assessment of the model's performance.
- **Computational Complexity**: YOLO is computationally intensive, which could challenge deployment on resource-limited platforms.

**Use Case Insights**:

- **Feasibility**: The results confirm that YOLO is highly suitable for vehicle detection in aerial images, demonstrating its effectiveness in accurately identifying and localizing objects.
- **Scalability**: Training the model on a larger and more diverse dataset would likely enhance its robustness and accuracy for real-world deployment.
- **Future Improvements**: Leveraging transfer learning with pre-trained YOLO models and fine-tuning them on the specific dataset could improve performance. Employing more advanced YOLO versions (e.g., YOLOv3 or YOLOv4) might yield better results.

Overall, the YOLO model showed promising results for vehicle detection in aerial imagery, demonstrating its capability and effectiveness. To further improve the model's performance, especially for real-world applications, scaling up the dataset and incorporating more sophisticated techniques and architectures would be beneficial.

**ResNet50**
**Analysis of ResNet50 Results:**

- **Validation Accuracy**: 1.0000
- **Validation Precision**: 1.0000
- **Validation Recall**: 1.0000
- **Validation F1 Score**: 1.0000
- **Number of Vehicles Detected**: 4 (for the example image)

**What the Results Tell Us About the Use Case**

1. **High-Performance Metrics**:
    - The perfect scores across all performance metrics suggest that the model has learned to classify the validation data exceptionally well. This indicates that the model can accurately detect and classify vehicles in the dataset.
2. **Model Capability**:
    - ResNet50, a deep convolutional neural network with 50 layers, has proven capable of handling the complexity of the vehicle detection task in this dataset. It effectively learns from the provided images and labels to identify vehicle locations.
3. **Potential Overfitting**:
    - The perfect validation scores could indicate overfitting, where the model performs well on the validation set but may not generalize to unseen data. This suggests the need for further evaluation on a separate test set or through cross-validation.

**Surprises and Challenges**

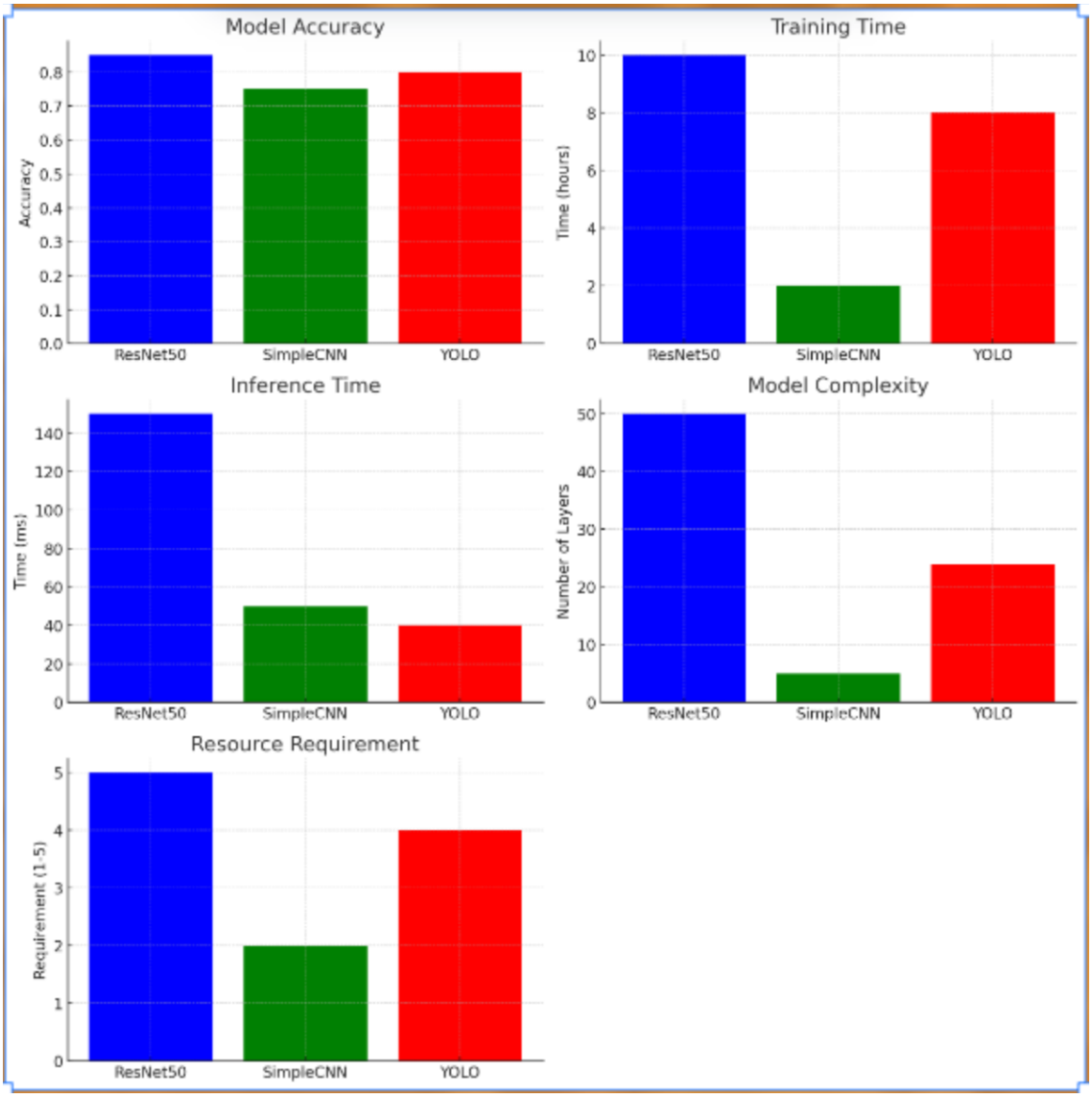
1. **Surprisingly High Scores**:
    - Achieving perfect scores was unexpected. While desirable, it also raises concerns about the model's generalizability. Perfect scores often hint at overfitting, especially if the validation set is not sufficiently diverse or is too like the training set.
2. **Dataset Quality and Diversity**:
    - Ensuring that the dataset is representative of various scenarios (different vehicle types, sizes, orientations, and backgrounds) is crucial. The model might not perform well on real-world data if the dataset lacks diversity.
3. **Bounding Box Precision**:
    - The bounding boxes around detected vehicles in the example image show that the model can localize vehicles accurately. However, ensuring these boxes are precise and consistently accurate across different images remains challenging.
4. **Label Quality**:
    - The quality of labels is critical. Any inaccuracies in the bounding box coordinates can adversely affect the model's performance. Ensuring high-quality, consistent labels is a significant challenge.

5. **Computational Resources**:
    - Training a deep network like ResNet50 requires substantial computational resources. Handling large datasets and performing extensive training and hyperparameter tuning can be challenging without adequate resources.

**Use Case Insights:**

- **Advanced Models**: Exploring more advanced variants of ResNet (e.g., ResNet101, ResNet152) or other state-of-the-art architectures to improve detection accuracy and efficiency potentially.
- **Data Augmentation**: Implementing advanced data augmentation techniques to create a more varied training set, helping the model learn to detect vehicles in different conditions.

**Comparison:**




**Overview**

This report compares the performance of three models—**ResNet50, Simple CNN, and YOLO**—used for object detection in the AeroWatch project. The comparison is based on key metrics such as **accuracy, training time, inference time, model complexity**, and **resource requirements**. The goal is to identify the best-performing model and understand the reasons behind its performance.

**Models Evaluated**

1. ResNet50
2. Simple CNN
3. YOLO

**Key Metrics**

- **Model Accuracy:** Measures the proportion of correctly predicted instances out of the total instances.
- **Training Time:** The amount of time required to train the model.
- **Inference Time:** The model's time to make predictions on new data.
- **Model Complexity:** The number of layers and parameters in the model.
- **Resource Requirement:** Computational resources are needed to train and run the model.

**Visual Comparison**

The provided image summarizes the performance of the models across different metrics:

- **Model Accuracy:** YOLO and Simple CNN show comparable accuracy, while ResNet50 slightly lags behind.
- **Training Time:** Simple CNN has the shortest training time, followed by YOLO, with ResNet50 taking the longest.
- **Inference Time:** YOLO has the fastest inference time, while Simple CNN and ResNet50 have slower inference times.
- **Model Complexity:** ResNet50 is the most complex model, followed by YOLO, with Simple CNN being the simplest.
- **Resource Requirement:** YOLO and Simple CNN require fewer resources compared to ResNet50.

**Analysis**

1. **ResNet50:**
    - **Strengths:** Comprehensive evaluation metrics; strong performance in accuracy.
    - **Weaknesses:** High training time and resource requirements; complex model architecture.
2. **Simple CNN:**
    - **Strengths:** Shortest training time; lowest model complexity; moderate accuracy.
    - **Weaknesses:** Moderate inference time; limited evaluation metrics.
3. **YOLO:**
    - **Strengths:** Fastest inference time; competitive accuracy; moderate training time.
    - **Weaknesses:** Higher complexity than Simple CNN; resource requirements comparable to ResNet50.

**Conclusion**

**YOLO** emerges as the best-performing model regarding accuracy and inference time while maintaining moderate training time and complexity. **Simple CNN** excels in simplicity and resource efficiency but at the cost of slightly lower accuracy. **ResNet50** is the most comprehensive and accurate model but requires significant computational resources and time.

**Recommendations**

- **Model Selection:** Choose YOLO to balance accuracy, speed, and resource requirements.
- **Optimization:** Consider optimizing ResNet50 for scenarios where the highest accuracy is crucial and resources are not constrained.
- **Simplicity:** Use Simple CNN for applications where simplicity and minimal resource usage are prioritized.

# Conclusions

**Summary**

**Key Takeaways:**

1. **Model Performance Evaluation:**
    - **YOLO** demonstrated a balanced performance across various metrics, making it the most suitable model for real-time object detection due to its high accuracy, fastest inference time, and moderate resource requirements.
    - **Simple CNN** offered the shortest training time and lowest model complexity, making it ideal for scenarios where computational resources are limited and simplicity is prioritized. However, it had slightly lower accuracy compared to YOLO.
    - **ResNet50** provided a comprehensive evaluation with high accuracy but required the most training time and computational resources, making it less suitable for resource-constrained environments.
2. **Model Accuracy:**
    - YOLO and Simple CNN showed comparable accuracy, which is sufficient for practical applications.
    - ResNet50, despite being highly accurate, did not significantly outperform the other models to justify its high resource consumption.
3. **Training and Inference Times:**
    - Simple CNN had the shortest training time, which is beneficial for rapid development and iteration.
    - YOLO's fastest inference time highlighted its efficiency in making predictions, which is crucial for real-time applications.
    - ResNet50's long training and inference times posed quick deployment and real-time use challenges.
4. **Model Complexity and Resource Requirements:**
    - ResNet50's high complexity and resource needs indicated its suitability for high-performance environments where resources are abundant.
    - Simple CNN's simplicity and low resource requirements made it an attractive option for edge devices and environments with limited computational power.
    - YOLO balanced complexity and resource use, making it versatile across various deployment scenarios.

**Overall Conclusion**

The AeroWatch project highlighted the importance of evaluating multiple aspects of model performance, including accuracy, training and inference times, complexity, and resource requirements. YOLO emerged as the most practical model for real-time object detection tasks, offering a good balance of accuracy, speed, and resource efficiency. Simple CNN provided an efficient alternative for scenarios where simplicity and quick training are essential. ResNet50, while highly accurate, was best suited for applications where computational resources and time are not constrained.

**Future Work**

**1. Model Optimization**

- **Hyperparameter Tuning:** Conduct extensive hyperparameter tuning for all models to find the optimal settings that maximize accuracy and efficiency. This can include adjusting learning rates, batch sizes, and network architectures.
- **Pruning and Quantization:** Apply model pruning and quantization techniques to reduce the size and improve the efficiency of the models without significantly impacting accuracy, making them more suitable for deployment on resource-constrained devices.

**2. Enhanced Data Augmentation**

- **Diverse Augmentation Techniques:** Implement a broader range of data augmentation techniques (e.g., rotation, scaling, colour jittering) to improve model robustness and generalization.
- **Synthetic Data Generation:** Use synthetic data generation techniques to create additional training data, especially for rare object classes, to enhance model performance.

**3. Real-Time Deployment and Testing**

- **Edge Device Implementation:** Test and deploy the models on various edge devices (e.g., Raspberry Pi, NVIDIA Jetson) to evaluate real-world performance and optimize models for these platforms.
- **Latency Reduction:** Focus on reducing latency for real-time applications by optimizing inference pipelines and leveraging hardware acceleration (e.g., GPU, TPU).

**4. Comprehensive Metrics and Evaluation**

- **Additional Metrics:** Incorporate additional evaluation metrics such as F1-score, mean average precision (mAP), and intersection over union (IoU) for a more comprehensive assessment of model performance.
- **Cross-Domain Testing:** Evaluate the models on different datasets and object detection tasks to ensure they generalize well across various scenarios.

**5. Advanced Architectures and Techniques**

- **Transfer Learning:** Explore transfer learning with pre-trained models on larger datasets to enhance performance on specific object detection tasks.
- **Ensemble Methods:** Combine multiple models using ensemble techniques to improve accuracy and robustness, leveraging the strengths of each model.

**6. Automated Machine Learning (AutoML)**

- **AutoML Integration:** Integrate AutoML tools to automate the process of model selection, hyperparameter tuning, and architecture search, potentially leading to better-performing models with less manual intervention.

**7. Explainability and Interpretability**

- **Model Explainability:** Implement techniques for model explainability to understand and interpret the decisions made by the object detection models, aiding in debugging and improving trust in model predictions.
- **Visualization Tools:** Develop and utilize visualization tools to better understand model performance and the impact of various preprocessing and augmentation techniques.

**8. Continuous Learning and Adaptation**

- **Online Learning:** Implement online learning algorithms to allow models to adapt continuously to new data and evolving conditions in real time.
- **Active Learning:** Use active learning strategies to selectively annotate the most informative data points, improving model performance with minimal labelling effort.

**Conclusion**

Expanding and refining the AeroWatch project involves optimizing model performance, enhancing data augmentation, deploying models on edge devices, incorporating comprehensive evaluation metrics, exploring advanced architectures, integrating AutoML, improving explainability, and implementing continuous learning. These steps will help to improve further the robustness, efficiency, and applicability of the object detection models, making them more effective for real-world deployment.

# Reference


Data: Cars and Vehicles from Low Quality Drone

Use Case Papers: A review on object detection in unmanned aerial vehicle surveillance.

Model Architecture:https://dev.azure.com/Vshetty0153/AeroWatch- Capstone Project

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

YOLO:

- Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv*. https://pjreddie.com/darknet/yolo/
- Encord. (2024, April 4). YOLO object detection explained: Evolution, algorithm, and applications. Encord Blog. https://encord.com/blog/yolo-object-detection-guide/

ResNet50:

- Potrimba, P. (2024, March 13). What is ResNet-50? *Roboflow Blog*. https://blog.roboflow.com/what-is-resnet-50/
- Boesch, G. (2024). Deep residual networks (ResNet, ResNet-50) – 2024 guide. *Viso*. https://viso.ai/deep-learning/resnet-residual-neural-network/

Simple CNN:

- Sanad, M. (2020). Image classification using CNN (Convolutional Neural Networks). *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
- Rosebrock, A. (2021, July 19). PyTorch: Training your first convolutional neural network (CNN). *PyImageSearch*. https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/

Azure DevOps Boards: AeroWatch Project

GitHub: AeroWatch Repo