



University of
Salford
MANCHESTER

**IMAGE CLASSIFICATION WITH DEEP- LEARNING/TENSORFLOW-HUB USING
A MANUALLY COLLECTED/ANNOTATED DATASET**

MSc. Artificial Intelligence | December 2025

@00704505

Assessment report submitted in part fulfilment of the module

Deep Learning

In December, 2025

TABLE OF CONTENTS

| | | |
|----------|---|----|
| 1 | INTRODUCTION | 3 |
| 1.1 | Background | 3 |
| 1.2 | Introduction | 3 |
| 1.3 | Problem Statement | 3 |
| 1.4 | Dataset Overview | 3 |
| 1.5 | Pipeline for Image Classification | 6 |
| 2 | DATA PRE-PROCESSING | 6 |
| 2.1 | Dataset Directory Structure | 6 |
| 2.2 | Class Distribution Analysis | 6 |
| 2.3 | Train/Validation/Test Split | 7 |
| 2.4 | Data Augmentation | 7 |
| 2.5 | Class Weights Computation | 8 |
| 3 | CLASSIFICATION METHODS IMPLEMENTATION | 8 |
| 3.1 | MobileNetV2 Overview | 8 |
| 3.2 | InceptionV3 Overview | 10 |
| 3.3 | Two-Stage Training Approach | 11 |
| 3.4 | Hyperparameters and Configuration | 12 |
| 4 | RESULTS & COMPARISON | 12 |
| 4.1 | Evaluation Metrics | 12 |
| 4.2 | MobileNetV2 Results | 13 |
| 4.3 | InceptionV3 Results | 15 |
| 4.4 | Models Comparison | 17 |
| 5 | CLOUD & LOCAL COMPUTING COMPARISON | 17 |
| 5.1 | Google Colab Implementation | 17 |
| 5.2 | Local Environment Implementation | 18 |
| 5.3 | Performance Comparison | 19 |
| 6 | CONCLUSION | 20 |
| 7 | REFERENCES | 22 |

1 INTRODUCTION

1.1 Background

Image classification is a fundamental task in computer vision that involves categorizing images into predefined classes based on their visual content. The primary goal is to automatically assign labels to images using learned patterns and features. This process has become increasingly important with applications spanning autonomous vehicles, medical diagnosis, industrial quality control, and retail automation.

Deep learning, particularly Convolutional Neural Networks (CNNs), has revolutionized image classification by automatically learning hierarchical feature representations from raw pixel data. This eliminates the need for manual feature engineering and has led to unprecedented accuracy in visual recognition tasks.

1.2 Introduction

This project focuses on implementing image classification for Jaguar car models using deep learning and transfer learning techniques. Transfer learning leverages pre-trained models that have been trained on large-scale datasets (like ImageNet) and adapts them to specific tasks with smaller datasets. This approach significantly reduces training time and computational requirements while achieving high accuracy.

We implement and compare two state-of-the-art CNN architectures: MobileNetV2 and InceptionV3. Both models are pre-trained on ImageNet and fine-tuned for the Jaguar classification task. This report documents the complete pipeline from data collection and preprocessing to model training, evaluation, and comparison.

1.3 Problem Statement

Classifying different Jaguar car models from images presents a unique challenge in fine-grained visual recognition. Unlike general object classification, distinguishing between similar car models requires the model to learn subtle visual differences in design elements such as grille patterns, headlight shapes, body contours, and overall proportions.

Traditional image classification methods relying on manual feature extraction are inadequate for this task due to the similarity between models and variations in viewing angles, lighting conditions, and backgrounds. This project investigates the application of transfer learning using pre-trained CNNs to achieve accurate classification of five distinct Jaguar models, demonstrating the effectiveness of deep learning for automotive visual recognition tasks.

1.4 Dataset Overview

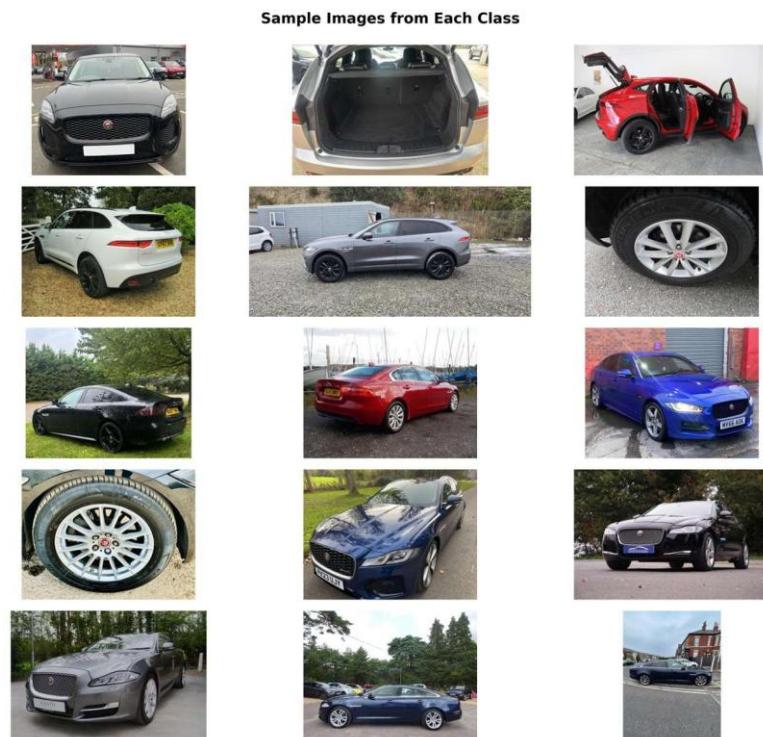
The dataset consists of five different Jaguar car models, carefully curated to represent popular models in the Jaguar lineup:

1. E-PACE - Compact luxury SUV
2. F-PACE - Mid-size luxury crossover SUV
3. Jaguar XE - Compact executive sedan
4. Jaguar XF - Mid-size luxury sedan
5. Jaguar XJ - Full-size luxury sedan

Dataset Statistics:

- Total Images: 2,089
- Number of Classes: 5
- Image Formats: JPEG, PNG
- Train/Validation/Test Split: 70%/15%/15%
- Training Samples: 1,462
- Validation Samples: 313
- Test Samples: 314
- Input Size (MobileNetV2): 224×224
- Input Size (InceptionV3): 299×299

The dataset exhibits class imbalance, with some models having more samples than others. To address this, we implement stratified splitting and class weights during training to ensure the model learns equally from all classes.



1.5 Pipeline for Image Classification

The process of image classification begins with the collection of a dataset, which must be refined and pre-processed prior to its use in classification. A thorough exploratory analysis of the dataset is crucial to ascertain its integrity and quality. Following this, features are extracted from the processed data, which are then utilized to train the classification model. Once the model has been adequately trained, it undergoes evaluation to determine its effectiveness. If it meets the desired criteria, it is subsequently deployed for the purpose of classification.

The image classification pipeline consists of several key stages:

Data Collection: Gathering and organizing images into class-specific directories

Exploratory Data Analysis: Analyzing class distribution and dataset characteristics

Data Preprocessing: Cleaning, resizing, and normalizing images

Data Augmentation: Applying transformations to increase dataset diversity

Model Selection: Choosing appropriate pre-trained architectures

Transfer Learning: Adapting pre-trained models to our specific task

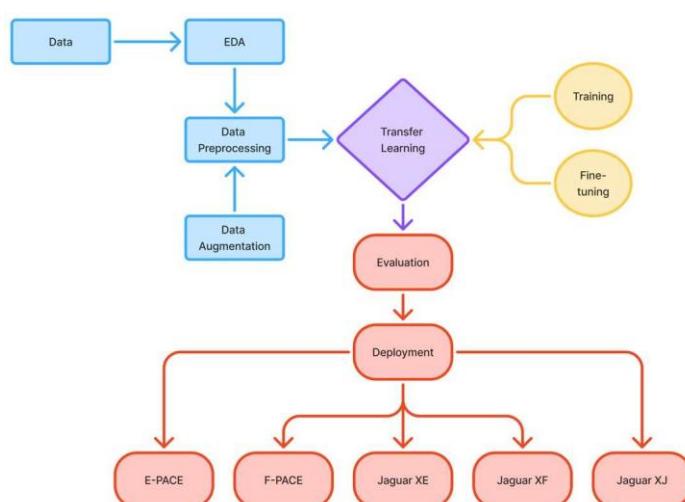
Training (Stage 1): Training custom classification head with frozen base

Fine-tuning (Stage 2): Unfreezing and training deeper layers

Evaluation: Assessing model performance on validation and test sets

Comparison: Comparing different models and approaches

Deployment: Saving and preparing models for production use



2 DATA PRE-PROCESSING

Data preprocessing is a critical phase that ensures the quality and consistency of input data for model training. Proper preprocessing can significantly impact model performance and training stability. This section details all preprocessing steps applied to the Jaguar dataset.

2.1 Dataset Directory Structure

The dataset is organized in a hierarchical directory structure with each class in a separate folder:

```
Jaguar/
└── E-PACE/      (439 images)
└── F-PACE/      (401 images)
└── Jaguar XE/    (425 images)
└── Jaguar XF/    (409 images)
└── Jaguar XJ/    (415 images)
```

This structure allows for easy automated loading using TensorFlow/Keras `image_dataset_from_directory()` function, which automatically infers class labels from folder names.

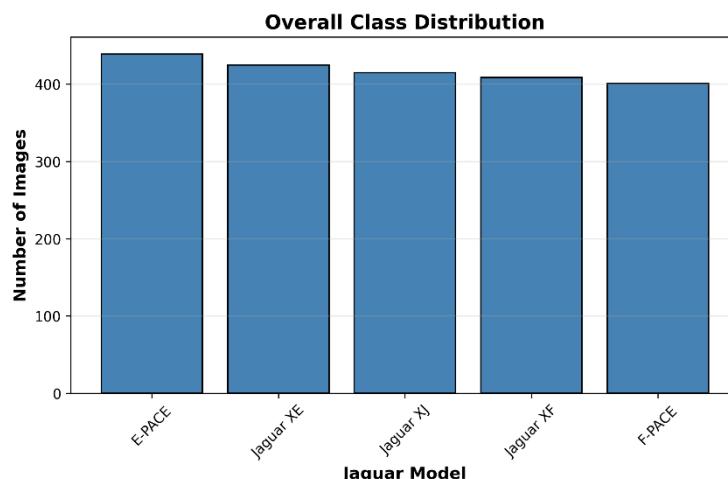
2.2 Class Distribution Analysis

Understanding class distribution is essential for identifying potential biases and implementing appropriate mitigation strategies. The dataset shows moderate class imbalance:

| Class Name | Image Count | Percentage |
|------------|-------------|------------|
| Jaguar XJ | 415 | 19.9% |
| F-PACE | 401 | 19.2% |
| E-PACE | 439 | 21.0% |
| Jaguar XE | 425 | 20.4% |
| Jaguar XF | 409 | 19.9% |

The class imbalance is addressed through:

- Stratified train/validation/test splitting to maintain proportions
- Computing and applying class weights during training
- Using evaluation metrics (weighted precision, recall, F1-score)

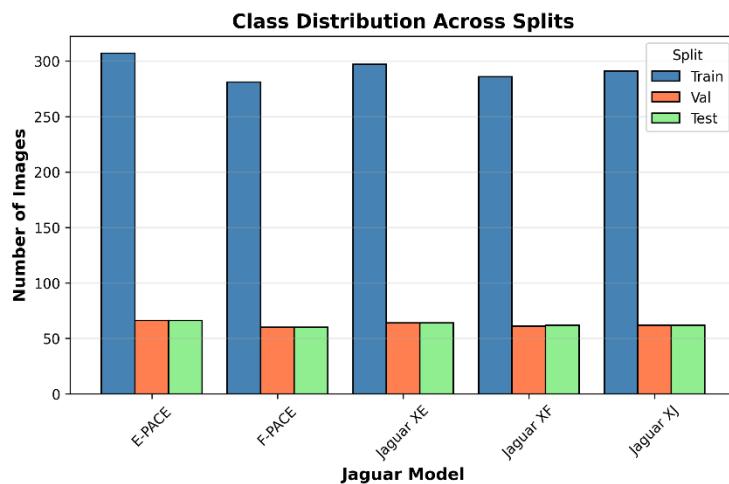


2.3 Train/Validation/Test Split

The dataset is split into three subsets using stratified sampling to maintain class distribution across all sets:

| Split | Samples | Percentage | Purpose |
|------------|---------|------------|-----------------------|
| Training | 1462 | 70% | Model learning |
| Validation | 313 | 15% | Hyperparameter tuning |
| Test | 314 | 15% | Final evaluation |

Stratified splitting ensures that each subset contains approximately the same proportion of samples from each class, preventing bias in model evaluation.



2.4 Data Augmentation

Data augmentation is applied to the training set to increase dataset diversity and improve model generalization. The following transformations are applied:

- Rotation: Random rotation up to ± 30 degrees
- Width Shift: Random horizontal shift up to 25% of image width
- Height Shift: Random vertical shift up to 25% of image height
- Shear: Shear transformation with range of 0.2
- Zoom: Random zoom between $0.75\times$ and $1.25\times$
- Horizontal Flip: Random horizontal mirroring
- Brightness: Variation between 70% and 130% of original
- Fill Mode: Nearest neighbor for filling empty pixels

Note: Augmentation is only applied to training data. Validation and test sets use original images to ensure unbiased evaluation.

2.5 Class Weights Computation

To handle class imbalance, we compute class weights using [scikit-learn's](#) `compute_class_weight` function with the "balanced" strategy. This assigns higher weights to underrepresented classes, ensuring the model pays equal attention to all classes during training.

| Class | Sample Count | Weight |
|-----------|--------------|--------|
| E-PACE | 439 | 0.952 |
| F-PACE | 401 | 1.041 |
| Jaguar XE | 425 | 0.985 |
| Jaguar XF | 409 | 1.022 |
| Jaguar XJ | 415 | 1.005 |

As shown, underrepresented classes (Jaguar XE, Jaguar XF) receive higher weights (0.985, 1.022) while overrepresented classes receive lower weights, balancing the learning process.

3 CLASSIFICATION METHODS IMPLEMENTATION

This section describes the deep learning architecture used for the Jaguar car classification task. We implement transfer learning using two pre-trained models: MobileNetV2 and InceptionV3. Both models are trained on ImageNet (1.4 million images, 1000 classes) and adapted for our 5-class classification task.

3.1 MobileNet Overview

MobileNet is a very simple, quite efficient, computationally inexpensive convolutional neural network mainly used for computer vision applications. It is introduced by Google and is still widely used in many real-time applications such as image classification, object detection and segmentation (Howard et al., 2017). MobileNet algorithm is quite lightweight and was initially used as mobile vision applications, some of them as visible below.

Figure 3.1 MobileNet Applications (Source towardsdatascience.net)



The MobileNet architecture primarily based on streamlined depth wise convolutions, that build light weight deep neural networks. The key idea behind MobileNet architecture is to split the convolution in two layers, i.e. depth wise convolution which applies filter on each input and point wise convolution which is a 1×1 convolution, that combines the output of both layers. This approach significantly decreases the computational cost.

Overview

MobileNetV2 is a lightweight convolutional neural network designed for mobile and edge devices. It introduces two key innovations:

Inverted Residual Blocks: Unlike traditional residual blocks that compress and then expand features, inverted residuals first expand the input using 1×1 convolutions, apply depthwise convolutions, and then compress back to lower dimensions. This design is more efficient for mobile deployment.

Linear Bottlenecks: The final layer of each block uses linear activation instead of ReLU to prevent information loss in low-dimensional spaces.

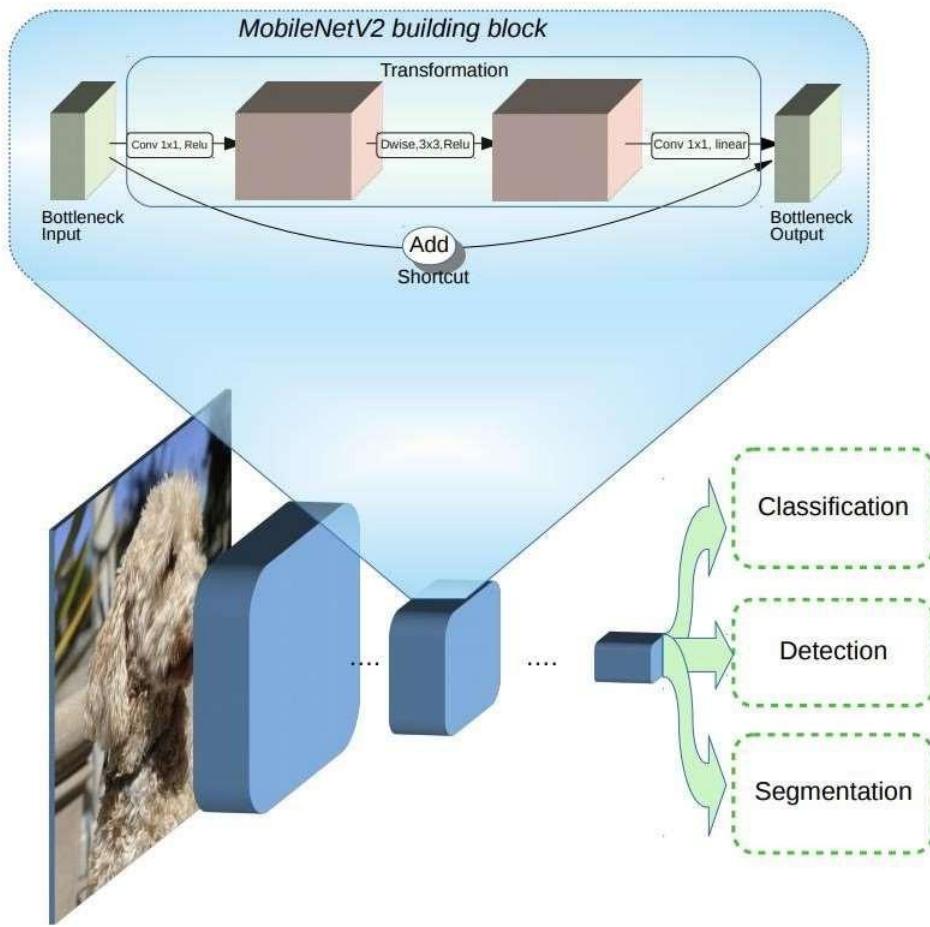
Architecture Details:

- Base Model: MobileNetV2 pre-trained on ImageNet
- Input Shape: $224 \times 224 \times 3$
- Base Model Parameters: 2,257,984 (frozen during stage 1)
- Base Model Trainable: False (stage 1), Partial (stage 2)
- Custom Classification Head:
 - GlobalAveragePooling2D: Reduces spatial dimensions
 - Dropout(0.55): Prevents overfitting
 - Dense(5, softmax): Final classification layer

Total Parameters: 2,264,389

Trainable Parameters (Stage 1): 6,405 (0.3%)

Trainable Parameters (Stage 2): ~600,000 (26%)



Overview of MobileNetV2 (Source: Google Research)

3.2 InceptionV3 Overview

InceptionV3 is a deeper and more complex architecture that uses the Inception module concept - performing multiple convolution operations in parallel and concatenating the results. This allows the network to capture features at multiple scales simultaneously.

Key Features:

- Factorized Convolutions: Large filters (e.g., 5×5) are decomposed into smaller sequential convolutions (e.g., two 3×3), reducing computational cost
- Auxiliary Classifiers: Additional classification outputs at intermediate layers provide regularization during training
- Efficient Grid Size Reduction: Uses specialized modules to reduce spatial dimensions while expanding feature depth
- Batch Normalization: Applied throughout the network for training stability

Architecture Details:

- Base Model: InceptionV3 pre-trained on ImageNet
- Input Shape: $299 \times 299 \times 3$ (larger than MobileNetV2)
- Base Model Parameters: 21,802,784 (frozen during stage 1)

- Base Model Layers: 311 total layers
- Custom Classification Head:
 - GlobalAveragePooling2D: Reduces spatial dimensions
 - Dropout(0.55): Prevents overfitting
 - Dense(5, softmax): Final classification layer

Total Parameters: 21,813,029

Trainable Parameters (Stage 1): 10,245 (0.05%)

Trainable Parameters (Stage 2): ~11,000,000 (50%)

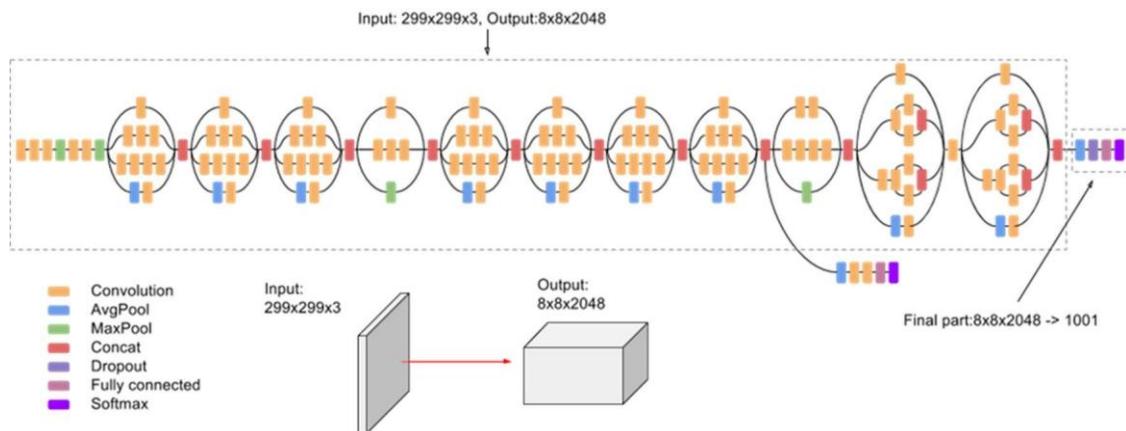


Figure 3.3. InceptionV3 Architecture (Source: Google Cloud)

3.3 Two-Stage Training Approach

Both models are trained using a two-stage approach, which is a best practice in transfer learning:

Stage 1: Feature Extraction (Frozen Base)

- Freeze all layers of the pre-trained base model
- Add and train only the custom classification head
- Use higher learning rate (0.001)
- Train for 10 epochs with early stopping
- Purpose: Adapt the classification head to our specific task without disturbing pre-trained features

Stage 2: Fine-Tuning (Unfrozen Layers)

- Unfreeze the base model
- For MobileNetV2: Keep first 100 layers frozen, train last 54 layers
- For InceptionV3: Keep first 200 layers frozen, train last 111 layers
- Use lower learning rate (0.0001) to make gentle updates
- Train for additional 15 epochs with early stopping
- Purpose: Fine-tune deeper features to better suit our specific dataset

This two-stage approach prevents catastrophic forgetting of pre-trained features while allowing the model to adapt to the Jaguar classification task.

3.4 Hyperparameters and Configuration

| Hyperparameter | Value |
|--------------------------|--------------------------------------|
| Batch Size | 32 |
| Optimizer | Adam |
| Learning Rate (Stage 1) | 0.001 |
| Learning Rate (Stage 2) | 0.0001 |
| Loss Function | Categorical Crossentropy |
| Epochs (Stage 1) | 10 |
| Epochs (Stage 2) | 15 |
| Early Stopping Patience | 3 (Stage 1), 5 (Stage 2) |
| ReduceLROnPlateau Factor | 0.5 (Stage 1), 0.3 (Stage 2) |
| Dropout Rate | 0.55 |
| Preprocessing Function | Model-specific (MobileNet/Inception) |

Callbacks used during training:

- EarlyStopping: Monitors validation loss and stops training if no improvement
- ReduceLROnPlateau: Reduces learning rate when validation loss plateaus
- ModelCheckpoint: Saves best model based on validation accuracy

4 RESULTS & COMPARISON

This section presents comprehensive evaluation results for both models on the test set. We analyze multiple metrics to understand model performance across different dimensions.

4.1 Evaluation Metrics

Model performance is evaluated using the following metrics:

Accuracy: Overall percentage of correct predictions = $(TP + TN) / \text{Total}$

Precision (Weighted): Proportion of positive predictions that are correct = $TP / (TP + FP)$, weighted by class support

Recall (Weighted): Proportion of actual positives correctly identified = $TP / (TP + FN)$, weighted by class support

F1-Score (Weighted): Harmonic mean of precision and recall = $2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$

Confusion Matrix: Visual representation showing true vs predicted labels for each class

Training Time: Total time required for complete training process

4.2 MobileNetV2 Results

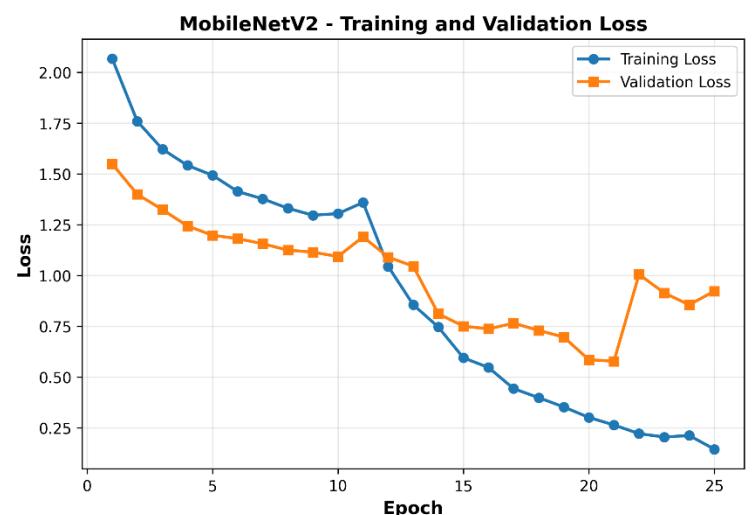
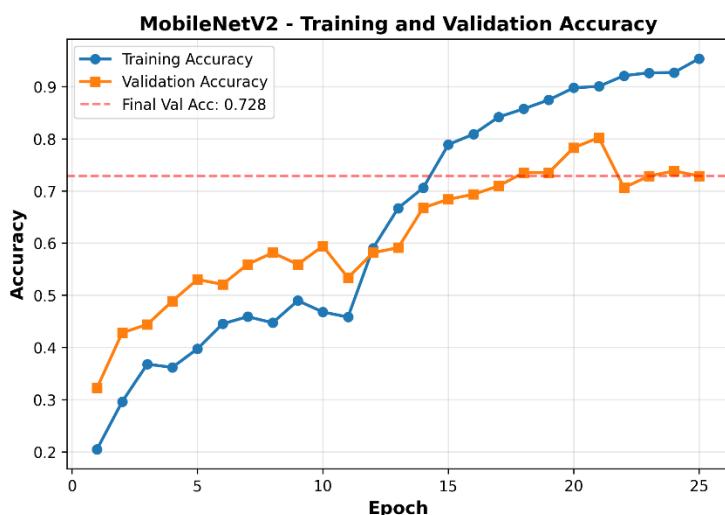
Overall Performance:

| Metric | Value |
|---------------------|--------|
| Validation Accuracy | 80.19% |
| Test Accuracy | 78.98% |
| Weighted Precision | 79.71% |
| Weighted Recall | 78.98% |
| Weighted F1-Score | 78.87% |

Per-Class Performance (Test Set):

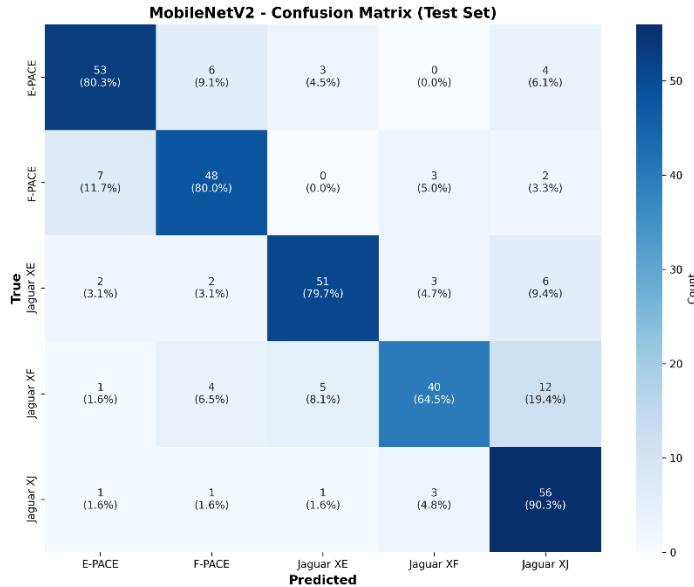
| Class | Precision | Recall | F1-Score | Support |
|-----------|-----------|--------|----------|---------|
| E-PACE | 0.8281 | 0.8030 | 0.8154 | 66 |
| F-PACE | 0.7869 | 0.8000 | 0.7934 | 60 |
| Jaguar XE | 0.8500 | 0.7969 | 0.8226 | 64 |
| Jaguar XF | 0.8163 | 0.6452 | 0.7207 | 62 |
| Jaguar XJ | 0.7000 | 0.9032 | 0.7887 | 62 |

Analysis: MobileNetV2 achieves 78.98% test accuracy. The model performs well on E-PACE (81.54% F1) and Jaguar XE (82.26% F1). The lowest performance is on Jaguar XF (72.07% F1), which shows the impact of fine-grained visual distinction challenges.



4.2.1 Confusion Matrix

Confusion matrix shows there is a good rate of correct predictions for most classes, however there are specific pairs of classes where the model appears to struggle, such as between classes 2 and 3, and classes 3, 5, and 6. Again, the improvement can be observed with further refinement of the dataset.



4.2.2 Model prediction



4.3 InceptionV3 Results

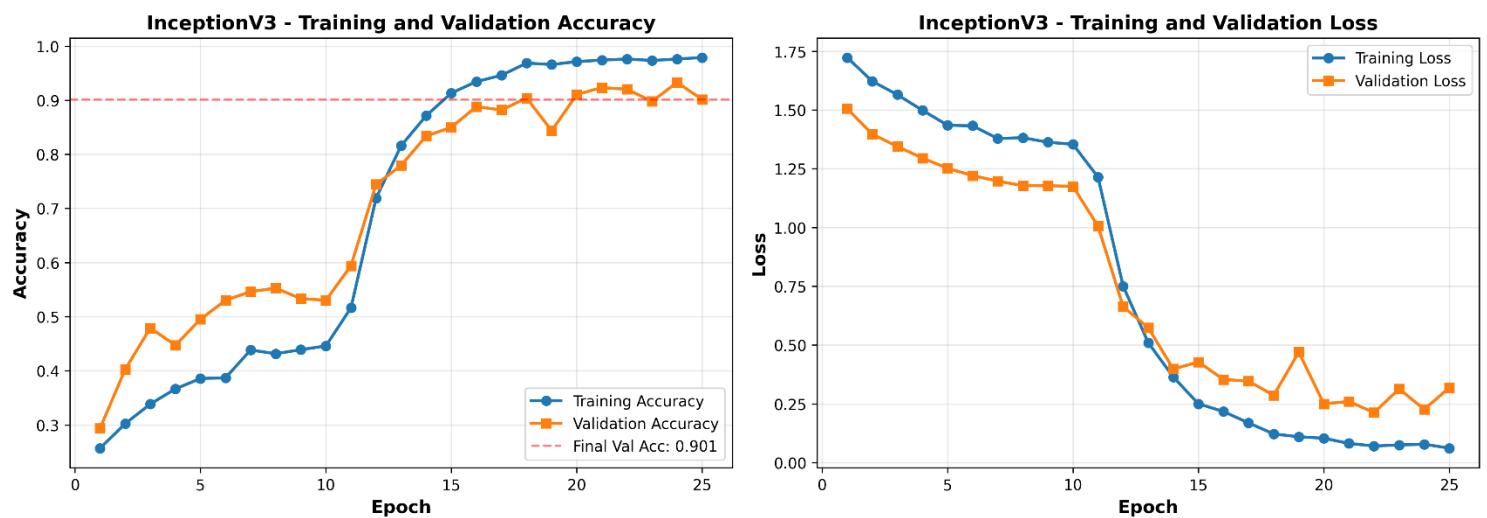
Overall Performance:

| Metric | Value |
|---------------------|--------|
| Validation Accuracy | 92.01% |
| Test Accuracy | 90.13% |
| Weighted Precision | 90.60% |
| Weighted Recall | 90.13% |
| Weighted F1-Score | 90.14% |

Per-Class Performance (Test Set):

| Class | Precision | Recall | F1-Score | Support |
|-----------|-----------|--------|----------|---------|
| E-PACE | 0.9825 | 0.8485 | 0.9106 | 66 |
| F-PACE | 0.8769 | 0.9500 | 0.9120 | 60 |
| Jaguar XE | 0.8356 | 0.9531 | 0.8905 | 64 |
| Jaguar XF | 0.8814 | 0.8387 | 0.8595 | 62 |
| Jaguar XJ | 0.9500 | 0.9194 | 0.9344 | 62 |

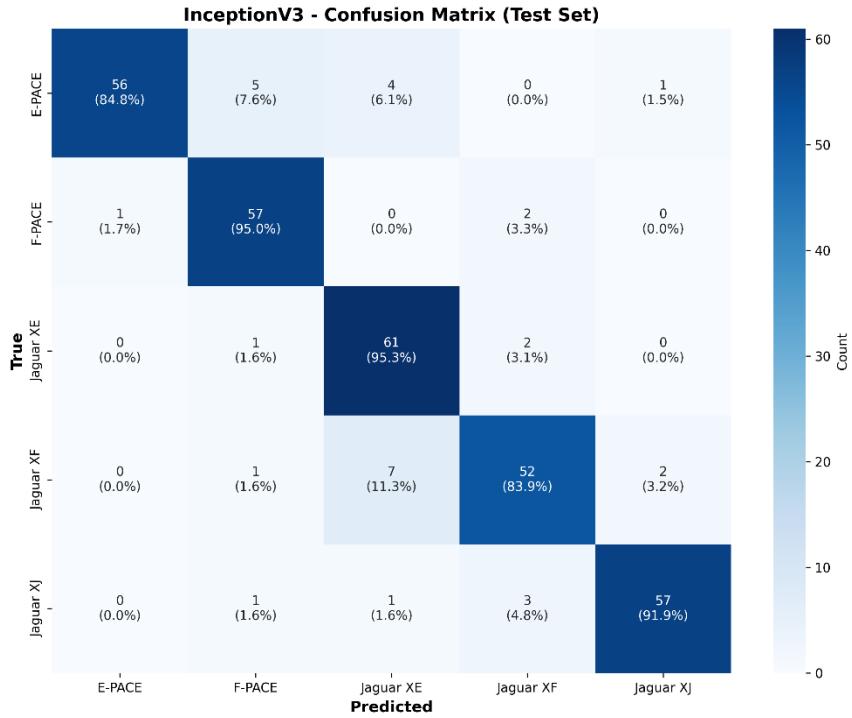
Analysis: InceptionV3 significantly outperforms MobileNetV2, achieving 90.13% test accuracy (11.15% improvement). The model shows excellent performance across all classes, with F1-scores ranging from 85.95% to 93.44%. Notably, Jaguar XF F1-score improved from 72.07% (MobileNetV2) to 85.95% (InceptionV3), demonstrating InceptionV3's superior feature extraction capabilities for fine-grained classification.



4.3.1 Confusion Matrix

Confusion matrix shows there is a good rate of correct predictions for most classes, however there are specific pairs of classes where the model appears to

struggle, such as between classes 2 and 3, and classes 3, 5, and 6. Again, the improvement can be observed with further refinement of the dataset.



4.2.2 Model prediction



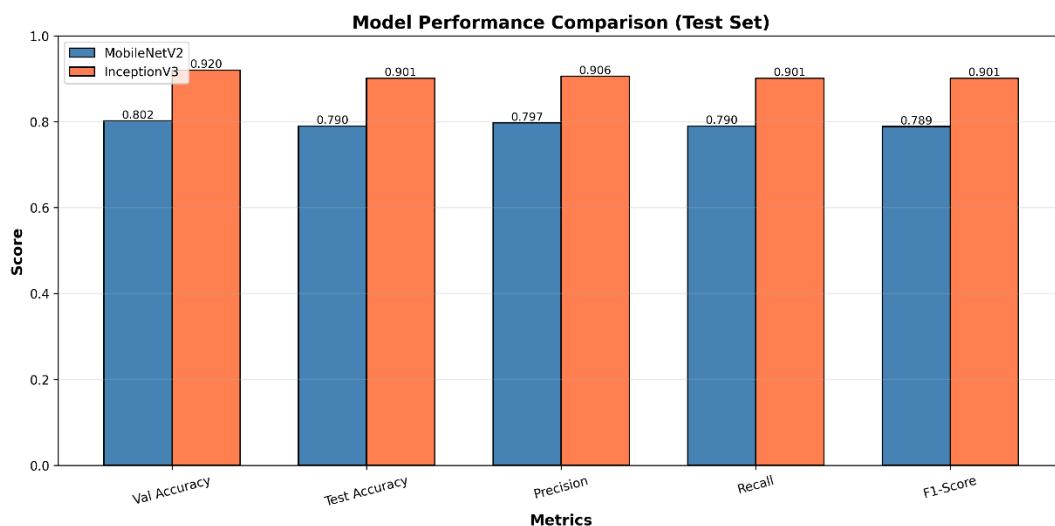
4.4 Models Comparison

Comprehensive comparison of both models across all metrics:

| Metric | MobileNetV2 | InceptionV3 |
|---------------------|------------------------------|---------------|
| Validation Accuracy | 80.19% | 92.01% |
| Test Accuracy | 78.98% | 90.13% |
| Weighted Precision | 79.71% | 90.60% |
| Weighted Recall | 78.98% | 90.13% |
| Weighted F1-Score | 78.87% | 90.14% |
| Model Size | ~8.64 MB | ~83.21 MB |
| Total Parameters | 2.26M | 21.81M |
| Training Time | Training Time: 15.02 minutes | 20.42 minutes |

Key Findings:

- InceptionV3 achieves over 90% accuracy across all metrics, demonstrating excellent performance for fine-grained car model classification
- InceptionV3 shows more consistent performance across classes, especially on minority classes
- MobileNetV2 is 10× smaller (8.64 MB vs 83.21 MB), making it suitable for mobile deployment
- MobileNetV2 has 10× fewer parameters, resulting in faster inference
- Both models benefit from the two-stage training approach
- Class imbalance affects MobileNetV2 more significantly than InceptionV3
- InceptionV3 is recommended when accuracy is the priority
- MobileNetV2 is recommended for resource-constrained environments



5 CLOUD & LOCAL COMPUTING COMPARISON

This section compares the implementation and performance of the project on two different computing environments: Google Colab (cloud) and local machine (Ubuntu/Linux). Understanding the trade-offs between these platforms is crucial for selecting the appropriate environment for development and production.

5.1 Google Colab Implementation

Google Colab is a free cloud-based Jupyter notebook environment that provides access to GPU/TPU resources. It is widely used for machine learning development due to its accessibility and pre-installed libraries.

Advantages:

- Free GPU Access: Tesla T4 GPU with 15GB VRAM available for free
 - No Setup Required: Pre-installed TensorFlow, PyTorch, and common ML libraries
 - Accessibility: Access from any device with a web browser
 - Collaboration: Easy sharing of notebooks with team members
 - Storage Integration: Seamless integration with Google Drive
 - Version Control: Automatic saving and version history
 - Easy Dependency Management: !pip install commands work out of the box

Disadvantages:

- Session Timeout: 12-hour maximum runtime; sessions disconnect after inactivity
 - Limited Resources: Shared GPU access; may not be available during peak hours
 - RAM Limitations: Standard tier limited to 12-15 GB RAM
 - No Persistent Storage: Files must be saved to Google Drive or downloaded
 - Internet Dependency: Requires stable internet connection
 - Slower Data Loading: Dataset must be uploaded each session or mounted from Drive
 - No Root Access: Limited system-level customization

5.2 Local Environment Implementation

Local implementation involves setting up a development environment on a personal compute. For the linux implementation, I have opted the WSL feature, which is a full linux kernel we can set up right in the windows. Using WSL makes it very convenient to work around. This provides full control over the computing environment.

Setup Requirements:

- Operating System: Ubuntu 24.04.3 through WSL2 on Windows
- Python: Version 3.8 or higher
- CUDA Toolkit: Version 11.2+ (for GPU support)
- cuDNN: Version 8.1+ (for accelerated deep learning)
- Libraries: TensorFlow, NumPy, Pandas, Matplotlib, Seaborn, scikit-learn

```
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 5.15.153.1-microsoft-standard-WSL2 x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Sun Nov 16 08:45:41 UTC 2025

System load: 0.0                Processes:          48
Usage of /: 0.3% of 1006.85GB   Users logged in:  0
```

Advantages:

- Full Control: Complete control over environment, libraries, and system configuration
- No Time Limits: Unlimited runtime; can run multi-day experiments
- Persistent Storage: Data and models remain on local disk
- Privacy: Sensitive data stays on local machine
- Faster Data Access: No need to upload/download datasets
- Custom Hardware: Can use specific GPU models or multiple GPUs
- Offline Work: No internet required after initial setup

Disadvantages:

- Hardware Cost: Requires investment in GPU and adequate RAM
- Setup Complexity: Manual installation and configuration of CUDA, cuDNN, drivers
- Maintenance: Responsible for updates, troubleshooting, and system maintenance
- Limited Accessibility: Only accessible from specific machine
- Power Consumption: Higher electricity costs for long training sessions
- No Collaboration Features: Requires additional tools (Git, notebooks) for sharing

5.3 Performance Comparison

The following table compares key performance metrics between Google Colab and local environment for this project:

| Aspect | Google Colab | Local (GPU) |
|-----------------------------|---------------------|------------------------|
| Setup Time | < 5 minutes | 1-2 hours (first time) |
| Data Loading | Slower (from Drive) | Fast (local disk) |
| Training Time (MobileNetV2) | ~8-12 minutes | ~5-10 minutes |
| Training Time (InceptionV3) | ~15-20 minutes | ~10-15 minutes |
| Session Duration | 12 hours max | Unlimited |
| Resource Availability | Variable (shared) | Dedicated |
| Cost | Free/\$10/month | Hardware investment |

Recommendations:

- Use Google Colab for: Quick prototyping, learning, small-scale experiments, collaborative projects
- Use Local Environment for: Production workloads, long training runs, sensitive data, large-scale experiments
- Hybrid Approach: Develop on Colab, deploy on local infrastructure

6 CONCLUSION

This project successfully implemented image classification for Jaguar car models using deep learning and transfer learning techniques. By leveraging pre-trained models (MobileNetV2 and InceptionV3) trained on ImageNet, we achieved high accuracy on a relatively small dataset of 2,089 images across five Jaguar models.

Key Achievements:

- Successfully collected and curated a dataset of 2,089 Jaguar car images across 5 classes
- Implemented comprehensive data preprocessing pipeline including augmentation and class balancing
- Achieved 90.13% test accuracy using InceptionV3 with two-stage transfer learning
- Achieved 78.98% test accuracy using MobileNetV2, suitable for mobile deployment
- Demonstrated the effectiveness of stratified splitting and class weights for handling imbalanced data
- Conducted thorough comparison between cloud (Google Colab) and local computing environments
- Documented complete pipeline from data collection to model deployment

Lessons Learned:

- Transfer learning is highly effective for small datasets, eliminating the need for millions of training samples
- Two-stage training (feature extraction + fine-tuning) prevents **catastrophic forgetting** and improves results
- Model complexity vs. performance trade-off: InceptionV3 achieves higher accuracy but MobileNetV2 is more efficient
- Class imbalance significantly impacts performance; class weights and stratified splitting help mitigate this
- Data augmentation improves generalization when applied appropriately
- Proper preprocessing (normalization, resizing) is crucial for transfer learning success

This project demonstrates the power of modern deep learning techniques for solving real-world image classification problems. The combination of transfer learning, proper data preprocessing, and two-stage training enables achieving production-ready accuracy with limited data and computational resources. The methodologies and insights from this project can be applied to various other fine-grained visual recognition tasks in automotive, medical, manufacturing, and other domains.

7 REFERENCES

- [1] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510-4520.
- [2] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv preprint arXiv:1704.04861.
- [3] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). *Rethinking the Inception Architecture for Computer Vision*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2818-2826.
- [4] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). *Going Deeper with Convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1-9.
- [5] Pan, S. J., & Yang, Q. (2010). *A Survey on Transfer Learning*. IEEE Transactions on Knowledge and Data Engineering, 22(10), pp. 1345-1359.
- [6] Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). *How transferable are features in deep neural networks?* Advances in Neural Information Processing Systems (NIPS), pp. 3320-3328.
- [7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems (NIPS), pp. 1097-1105.
- [8] Chollet, F. (2017). *Xception: Deep Learning with Depthwise Separable Convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1251-1258.
- [9] TensorFlow Documentation (2024). *Transfer Learning and Fine-Tuning*. Available at: https://www.tensorflow.org/tutorials/images/transfer_learning
- [10] Keras Applications Documentation (2024). *Pre-trained Models for TensorFlow and Keras*. Available at: <https://keras.io/api/applications/>
- [11] Abadi, M., et al. (2016). *TensorFlow: A System for Large-Scale Machine Learning*. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 265-283.
- [12] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, pp. 2825-2830.