Deep Learning Workshop - Assignment 1

By: Roei Yehezkel, 213138787, Omri Vilan, 213479686

Introduction

In this assignment, we tackled the Stanford Cars dataset to develop a classification model that predicts car types using CNN architectures. Several strategies, including custom CNN models, pretrained models, K-fold cross-validation, and advanced augmentation techniques, were applied. Additionally, new class categories were introduced to test the extensibility of the model.

EDA

Dataset Overview

The dataset consists of images from the Stanford Cars Dataset, divided into training and testing sets. Each image is labeled with a class representing a specific car model. The dataset includes bounding box information for each image.

- **Training set size:** 8144 images
- **Testing set size:** 8041 images

Sample Images

To better understand the dataset, here are a few sample images from the training set, showing examples of visually similar and dissimilar car models.

Differencing between the 2 Audi models will be harder seperable than between each Audy compared to the Infinity model.

Training Class Distribution:

- Most populated class: **Class 118 with 68 images**
- Least populated class: **Class 135 with 24 images**

Testing Class Distribution:

- Most populated class: **Class 118 with 68 images**
- Least populated class: **Class 135 with 24 images**

Data Content

Each sample in the dataset is an image of a car, along with associated metadata:

1. **Image Information:**
    - **Dimensions:** Images have varying resolutions (e.g., width × height). Images will typically need to be resized to a consistent input size for model training (e.g., 224 × 224 pixels).
    - **Channels:** Each image has **3 channels (RGB)**, representing red, green, and blue color components.
    - **Bounding Boxes:** Each image is accompanied by bounding box coordinates (x1, y1, x2, y2) for cropping.
2. **Labels:**
    - **Classes:** The dataset contains **196 classes**, each representing a unique car model (e.g., Audi 100 Sedan 1994, Infiniti QX56 SUV 2011).
    - **Class Distribution:** The classes are imbalanced, with some classes having significantly fewer samples than others.

Preprocessing

Preprocessing is necessary to standardize and enhance the dataset for training. Below are the required preprocessing steps:

1. **Bounding Box Cropping:**
    - Use the provided bounding box coordinates to crop the images, focusing on the car itself and removing unnecessary background.
2. **Resizing:**
    - Images need to be resized to a consistent shape (e.g., 224 × 224) to ensure compatibility with most CNN architectures.
3. **Normalization:**

Normalize pixel values using the mean and standard deviation of the dataset

4. **Data Type Conversion:**
    - Convert images to tensors for use with PyTorch or other deep learning libraries.

Augmentation

Augmentation is essential, especially to combat class imbalance and improve generalization.
optional augmentations:

1. **Geometric Transformations:**
    - **Random Cropping:** Enhance the model's ability to recognize cars from partial or varying views.
    - **Random Resizing:** Resizing with scaling variations to simulate different viewing conditions.
    - **Horizontal Flipping:** Most cars are symmetrical, so flipping can effectively double the dataset size.
2. **Color Transformations:**
    - **Color Jitter:** Slightly modify brightness, contrast, and saturation to simulate different lighting conditions.
    - **Grayscale Conversion:** Occasionally convert images to grayscale to ensure the model doesn't overly rely on color.
3. **Affine Transformations:**
    - **Rotation:** Rotate images by small angles (e.g., ±10°) to simulate slight tilts.
    - **Translation:** Shift images horizontally or vertically.
4. **Noise and Blurring:**
    - **Gaussian Blur:** Add a slight blur to simulate out-of-focus images.
    - **Random Noise:** Introduce minimal noise to make the model robust to noisy real-world data.

Benchmark Comparison

We compared the results of custom CNN models and pretrained architectures. The following benchmarks represent pretrained models' performance as reference points.

Method	**Accuracy (%)**
ResNet34	**88**
ResNet50	**89**
VGG16	**84**
EfficientNet	**81**

Custom CNN Architecture

Transformations used are resizing and normalizing the images

Convolutional Layers

The custom CNN architecture comprised 4 convolutional blocks, each with:

- Convolutional layers for spatial feature extraction.
- Batch normalization for stabilizing training.
- ReLU activation to learn non-linear patterns.
- Max-pooling layers for dimensionality reduction.

Feature map depths increased as follows:

- Block 1: Input: 3 channels → Output: 64 feature maps.
- Block 2: Input: 64 → Output: 128.
- Block 3: Input: 128 → Output: 256.
- Block 4: Input: 256 → Output: 256.

Fully Connected Layers

Features extracted from convolutional layers were flattened and passed through fully connected layers for final classification:

1. **FC1**: Flattened features → 512 neurons.
2. **Output Layer**: 512 neurons → Number of classes.

Dropout (50%) and batch normalization were applied to prevent overfitting.

Results for Custom Model

Baseline Performance

- **Training Loss**: Improved across epochs.
- **Validation Accuracy**: Plateaued around **26.56%**, indicating overfitting.
- **Test Accuracy**: Mean accuracy of **27.09%**, confirming limited generalization.

Suggestions for Improvement

To address overfitting and improve accuracy, several modifications were applied:

1. Data Augmentation

- **Transformations**: Random cropping, flipping, color jitter, and rotation.
- **Impact**: Decreased validation accuracy to **5.5%** and test accuracy to **6.16%**.

2. Modified Architecture

- **Changes**: Reduced fully connected neurons to 128, added convolutional layers for better feature extraction, and used a smaller learning rate.
- **Impact**: Validation accuracy increased to **9.1%**, and test accuracy reached **9.26%**.

3.Regularization

- **Increased Dropout**: Increase dropout rates in the fully connected layers (e.g., from 50% to 60%) and add dropout layers after convolutional blocks to randomly deactivate neurons during training.
- **L2 Regularization**: Implement early stopping based on validation loss to terminate training when overfitting begins.

4. Inference-Time Augmentation (ITA)

- **Process**: Averaged predictions across multiple augmentations of the test dataset.
- **Impact**: Test accuracy increased to **30.02%.**

K-Fold Cross-Validation

To ensure robustness, 5-fold cross-validation was applied. The results were consistent across folds:

Fold	Validation Accuracy (%)	Test Accuracy (%)	Test Accuracy with ITA (%)
1	26.68	26.01	28.53
2	25.28	27.34	31.7
3	27.11	27.3	30.15
4	25.89	27	30.02
5	27.79	27.8	30.02

Adding a New Class

To evaluate extensibility, the **Toyota Tacoma** class was introduced with 100 images:

- **Train Dataset**: 80 images.
- **Test Dataset**: 20 images.

Results for the New Class:

- **Test Accuracy**: **85%**
- **Misclassified Images**: 3

Pretrained Models

Pretrained models such as **ResNet50**, **VGG16**, **DenseNet169** and **EfficientNet-B4** were fine-tuned using the Cars dataset.

Results

Model	num. parameters	Validation Loss	Validation Accuracy (%)	Test Loss	Test Accuracy (%)	# unique correct samples	# unique errors
VGG16	135,063,556	1.0085	71.15	1.0338	71.12	5,719	2,322
ResNet50	23,909,636	**0.4780**	86.49	0.4939	87.10	7,004	1,037
DenseNet169	12,810,820	0.4794	**86.86**	**0.4780**	**87.51**	**7,037**	**1,004**
EfficientNet-B4	17,900,044	0.9476	74.28	0.9468	73.87	5,940	2,101

Observations

1. Pretrained models outperformed the custom CNN in both accuracy and efficiency.
2. DenseNet169 showed the best performance with **87.45%** test accuracy.

Here is a conclusion of all experiments, including using the trained DenseNet169 model as a feature extractor and adding over it a logistic regression model. The combination performed almost as well as the base DenseNet model, but not quite as good, underperforming in all metrics (not by nuch).

Model Name	Precision	Recall	F1 Score	Testing Time (s)	Runtime (s)	# unique correct samples	# unique errors	Significant Changes/Notes & Parameter Settings
VGG16	0.7285	0.711	0.7101	47.26	919.69	5,719	2,322	Pretrained weights, unfreezing layers after epoch 4. Added augmentation, Adam optimizer (different LR for the classifier and the rest), LR scheduler(step size 5, gamma 0.5). batch size 32, 15 epochs
ResNet50	0.8754	0.871	0.8707	44.36	755.17	7,004	1,037	Pretrained weights, unfreezing layers after epoch 4. Added augmentation, Adam optimizer (different LR for the classifier and the rest), LR scheduler(step size 5, gamma 0.5). batch size 32, 15 epochs
DenseNet169	0.8805	0.875	0.8752	46.09	869.33	7,037	1,004	Pretrained weights, unfreezing layers after epoch 4. Added augmentation, Adam optimizer (different LR for the classifier and the rest), LR scheduler(step size 5, gamma 0.5). batch size 32, 15 epochs
EfficientNet_b4	0.7521	0.738	0.7377	45.38	1,045.59	5,940	2,101	Pretrained weights, unfreezing layers after epoch 4. Added augmentation, Adam optimizer (different LR for the classifier and the rest), LR scheduler(step size 5, gamma 0.5). batch size 32, 15 epochs
Logistic Regression	0.8551	0.853	0.8518	132.9		6884	1157	Trained DenseNet169 as a feature extractor with a Logistic Regression model (max_iter=1000) as the classifier.

Conclusion

1. **Custom Model Improvements**:

    - Data augmentation and architecture modification significantly decreases performance (from 22% to 9%, in terms of accuracy).
    - However, Inference-Time Augmentation improved performance to around 30% accuracy.

2. **Pretrained Models**:

    - Fine-tuning pretrained architectures demonstrated superior results.
    - DenseNet169 and ResNet50 outperformed EfficientNet and VGG16.

3. **Adding New Classes**:

    - The framework successfully extended to include new classes, achieving high accuracy for the **Toyota Tacoma** class.