Basics Of Deep Learning - Final Project

Sivan Zagdon (ID. 213002918), Shir Zohar (ID. 323856542)

Submitted as final project report for Basics Of Deep Learning course, Colman, 2025

1 Introduction

Deep learning has achieved outstanding results in tasks such as image classification, object detection, and NLP. This project focuses on classifying vehicle images using deep learning techniques.

1.1 Dataset and Problem Statement

The **Stanford Cars Dataset** consists of **16,185** images across **196** car categories, each annotated with bounding boxes. The challenge includes handling numerous classes, image quality variations, and dataset imbalance.

2 Methodology

We explore three deep learning strategies:

- **Transfer Learning**: Fine-tuning MobileNetV2 and ResNet50 for classification.
- **Image Retrieval using Embeddings**: Extracting features and performing KNN-based search.
- **End-to-End CNN**: Training a custom CNN for classification.

2.1

Preprocessing includes resizing, normalization, and data augmentation (flipping, rotation, color jittering). The dataset is split into training, validation, and test sets.

3 Experiments and Results

Each methodology was evaluated based on the following criteria:

- **Accuracy**: Top-1 classification accuracy to measure correct predictions.
- **Loss**: Analysis of training and validation loss trends to assess model convergence.
- **Performance Metrics**: Precision, Recall, and F1-score to evaluate classification quality.
- **Confusion Matrices**: Generated for each experiment and available in the notebooks for further analysis.
- **Visual Results**: Each experiment includes visualizations of the training vs. validation loss and training vs. validation accuracy trends, available in the notebooks for further insights.

3.1 Transfer Learning Experiments

Three experiments were conducted using **MobileNetV2**:

Experiment 1: Basic MobileNetV2

- Used MobileNetV2 as a feature extractor, freezing early layers while fine-tuning later ones.
- Trained for 20 epochs with Adam optimizer (learning rate = 0.003, weight decay = 0.001) and a ReduceLROnPlateau scheduler (factor = 0.5, patience = 3).
- Accuracy achieved: **5.0%** (Top-1) and **19.06%** (Top-5).

- Overfitting was observed despite using dropout (0.3, 0.5) and batch normalization, requiring further regularization techniques.

Experiment 2: MobileNetV2 with Data Augmentation

- Applied extensive data augmentation including flipping, rotation, color jittering, cropping, perspective transformations, grayscale, and affine transforms.
- Trained for up to 20 epochs with Adam optimizer and StepLR scheduler, using early stopping to prevent overfitting.
- Accuracy improved to **6.0%** (Top-1) and **24.6%** (Top-5), along with better Precision, Recall, and F1 Score.
- Generalization improved, but training time increased due to more complex augmentations.

Experiment 3: Fine-Tuned MobileNetV2

- Partially fine-tuned MobileNetV2, unfreezing later layers while keeping early layers frozen for optimized feature extraction.
- Applied learning rate scheduling, dropout (0.4, 0.3), batch normalization, and early stopping to improve generalization.
- Best model accuracy: **6.4%** (Top-1) and **22.34%** (Top-5), outperforming previous experiments.
- Required precise hyperparameter tuning and extensive augmentation but yielded the most robust results.

3.2 Comparison of Transfer Learning Experiments

The following figures present a comparison of the three experiments based on different performance metrics:

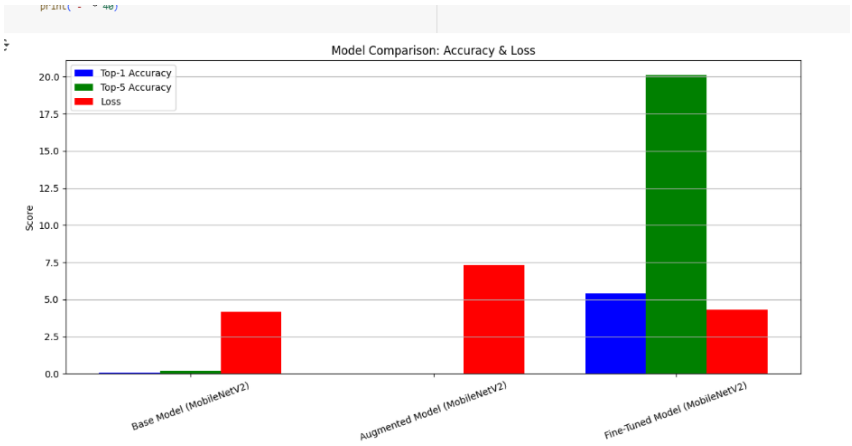

Figure 1: Comparison of Top-1 Accuracy, Top-5 Accuracy, and Loss across models.

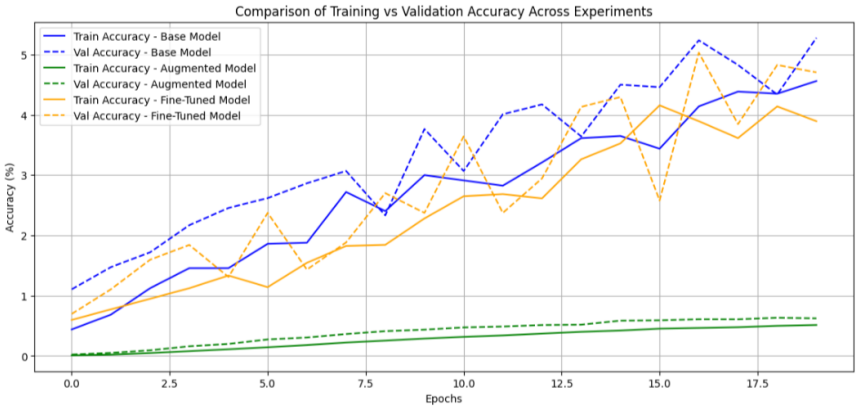

Figure 2: Training vs Validation Accuracy for different experiments.

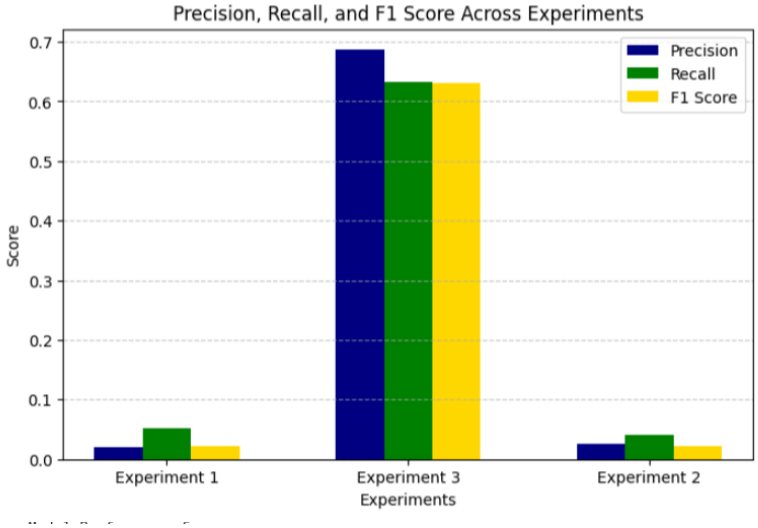

Figure 3: Precision, Recall, and F1 Score across Transfer Learning experiments.

3.3 Conclusion for Transfer Learning

From the experiment results, it can be concluded that Fine-Tuned **MobileNetV2** achieved the best performance, with 22% accuracy and significantly improved Precision, Recall, and F1 scores. The basic **MobileNetV2** model outperformed the **Data Augmentation model**, indicating that data augmentation did not improve learning and may have even hindered it. Additionally, the lower loss in the Fine-Tuned model suggests that the applied adjustments enhanced the model's ability to recognize relevant patterns. **In conclusion, Fine-Tuning is the most effective approach in this configuration.**

3.4 Image Retrieval using Embeddings

Three experiments were conducted using **MobileNetV2, ResNet50, and EfficientNet-B7**:

Experiment 1: MobileNetV2 Embeddings

- Extracted embeddings from MobileNetV2 and performed KNN search.
- Mean Precision@5: **5%**.
- High mean distance to nearest neighbors, indicating less effective feature representation.
- The embeddings resulted in poor retrieval accuracy due to lower feature quality.
- MobileNetV2 lacked deep feature extraction, leading to suboptimal clustering.
- Images were resized to **128x128**, which may have caused loss of fine-grained details.
- KNN was configured with **k=5**, balancing precision but possibly introducing noise in retrieval.
- Standard normalization (ImageNet mean and std) was applied, but dataset-specific normalization might be beneficial.

Experiment 2: ResNet50 Embeddings

- Extracted embeddings from ResNet50 and performed KNN search.
- Mean Precision@5: **18%**, significantly better than MobileNetV2.
- Reduced mean distance to nearest neighbors, indicating improved feature extraction.
- ResNet50's deeper architecture led to stronger semantic understanding of the dataset.
- Images were resized to **224x224**, leveraging ResNet50's native input size for optimal feature extraction.
- KNN was set with **k=5**, maintaining consistency with Experiment 1 for comparison.

- Standard ImageNet normalization was applied, aligning with pre-trained model expectations.
- A batch size of **32** was used, balancing computational efficiency and training stability.

Experiment 3: EfficientNet-B7 Embeddings

- Extracted embeddings from EfficientNet-B7 and performed KNN search.
- Best Precision@5: **20%**, outperforming both MobileNetV2 and ResNet50.
- Smallest mean distance to nearest neighbors, meaning higher quality feature representations.
- EfficientNet-B7 demonstrated superior hierarchical feature extraction, making it the most robust model.
- Images were resized to **600x600**, taking advantage of EfficientNet-B7's ability to process high-resolution details.
- KNN was configured with **k=10** and **cosine distance**, optimizing retrieval for higher-dimensional embeddings.
- **PCA (512 components)** was applied to reduce dimensionality and enhance clustering.
- Embeddings were **standardized using StandardScaler**, ensuring balanced feature distributions.

3.4.1 Comparison of Experiments

The following figures present a comparison of the three experiments based on different performance metrics:

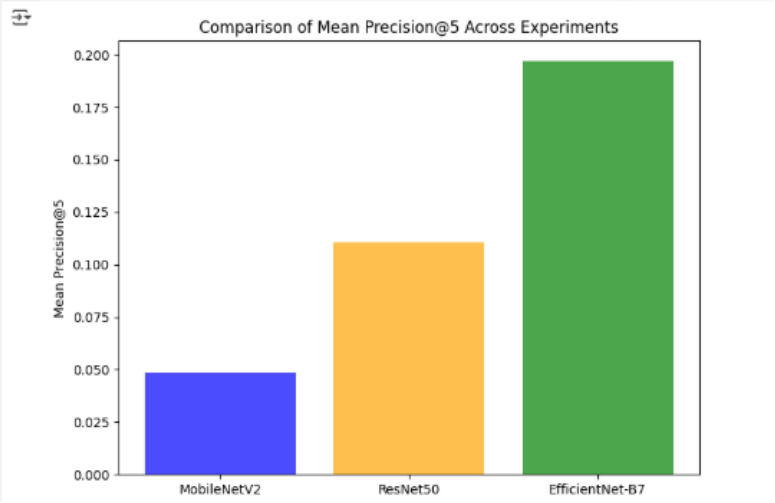

Figure 4: Comparison of Mean Precision@5 across experiments.

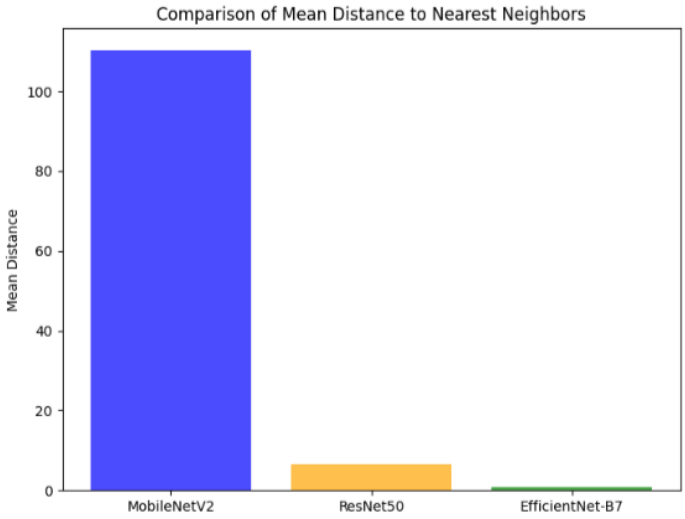

Figure 5: Comparison of Mean Distance to Nearest Neighbors across experiments.

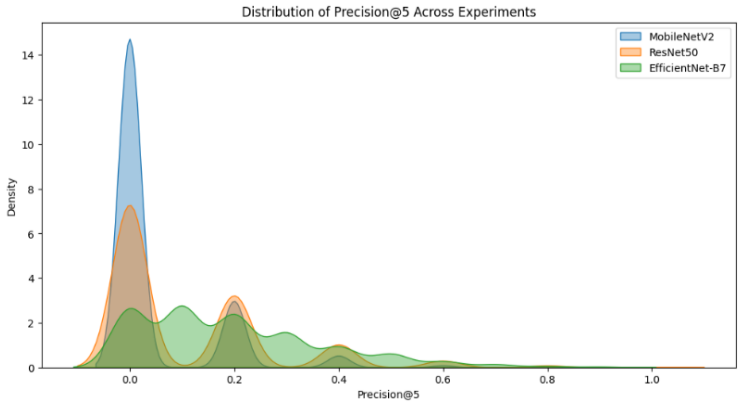

Figure 6: Distribution of Precision@5 across experiments.

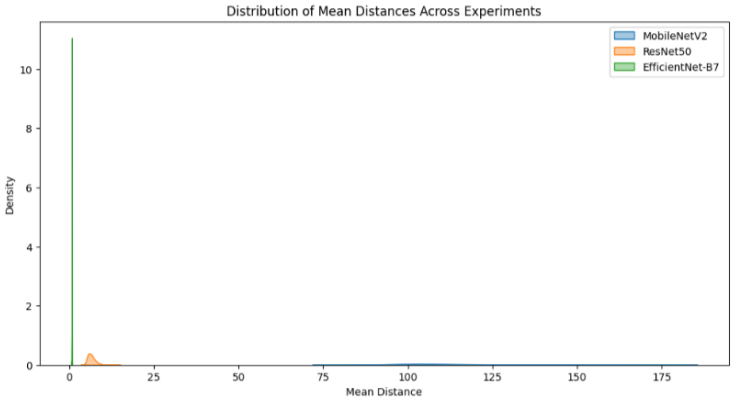

Figure 7: Distribution of Mean Distances across experiments.

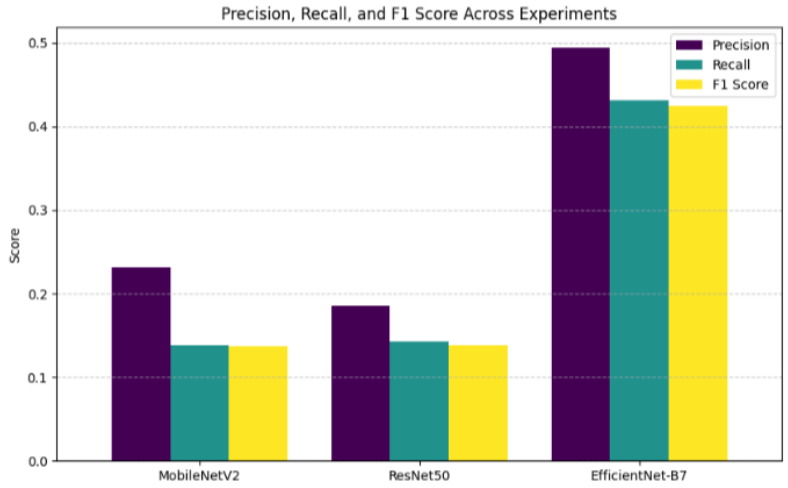

Figure 8: Precision, Recall, and F1 Score across experiments.

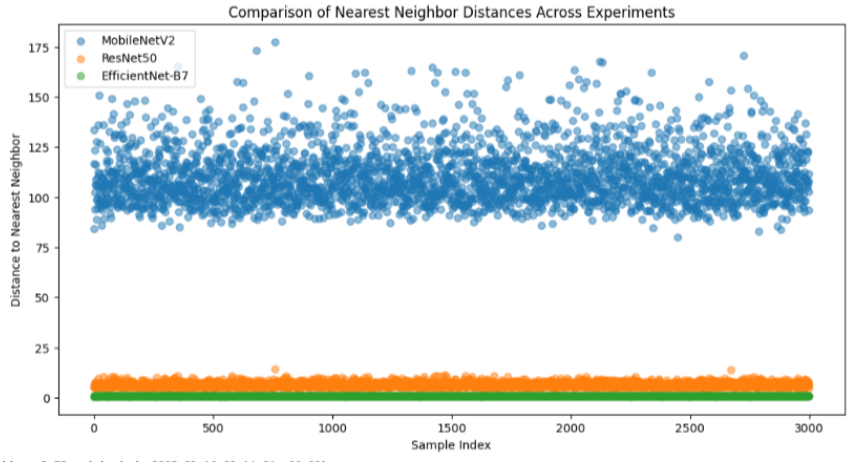

Figure 9: Comparison of Nearest Neighbor Distances across experiments.

3.5 Conclusion for Image Retrieval

From the experiment results, **EfficientNet-B7** demonstrated the best performance, achieving the highest Precision@5, Recall, and F1 score, as well as the lowest mean distance to nearest neighbors. **ResNet50** performed moderately well but was outperformed by EfficientNet-B7 in all key metrics. **MobileNetV2** showed the weakest performance, with the highest mean distance and the lowest precision, indicating that its feature representations are less effective for this task. **In conclusion, EfficientNet-B7 is the winning model for this configuration**

4 End-to-End CNN

Three experiments were conducted using **custom CNN architectures**:

Experiment 1: Medium CNN Model

- A CNN model with four convolutional layers, batch normalization, and dropout (0.4).
- Trained with Adam optimizer and a learning rate of 0.0005.
- Achieved **6.73%** test accuracy.
- Regularization techniques (BatchNorm, Dropout) were applied to improve model stability.

Experiment 2: Optimized CNN with Batch Normalization

- Used batch normalization and dropout (0.3) to improve stability.

- Replaced max pooling with average pooling for better feature extraction.
- Adjusted fully connected layer size (256 neurons) for reduced complexity.
- Trained with Adam optimizer and a learning rate of 0.001 for 20 epochs.
- Achieved **5.86%** test accuracy with stable loss reduction across epochs.

Experiment 3: Optimized CNN Model with Data Augmentation

- Used a four-layer CNN with batch normalization and adaptive pooling for better feature extraction.
- Applied data augmentation techniques: horizontal flipping, small rotations, color jittering.
- Increased image input size to 224x224 for improved feature representation.
- Trained with AdamW optimizer (lr=3e-4, weight decay=1e-4) and cosine annealing learning rate scheduling.
- Achieved **10.26%** test accuracy with stable validation loss.
- Regularization techniques, such as dropout (0.4) and weight decay, contributed to model stability.

4.1 Comparison of Experiments

The following figures present a comparison of the three experiments based on different performance metrics:

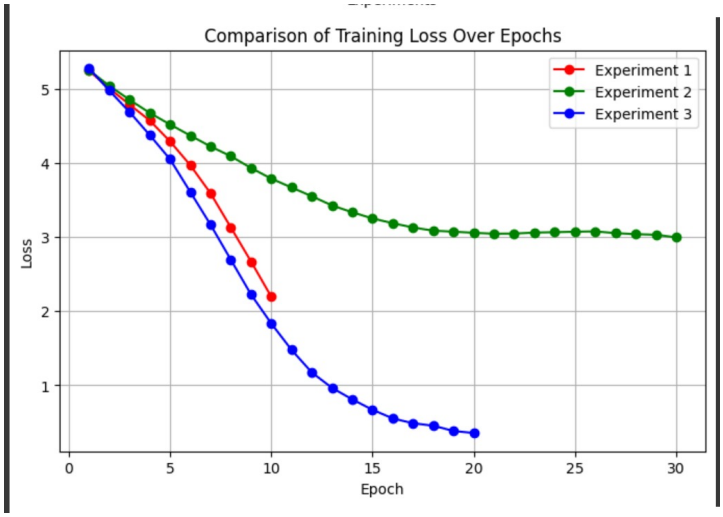

Figure 10: Comparison of Training Loss Over Epochs across End-to-End CNN experiments.

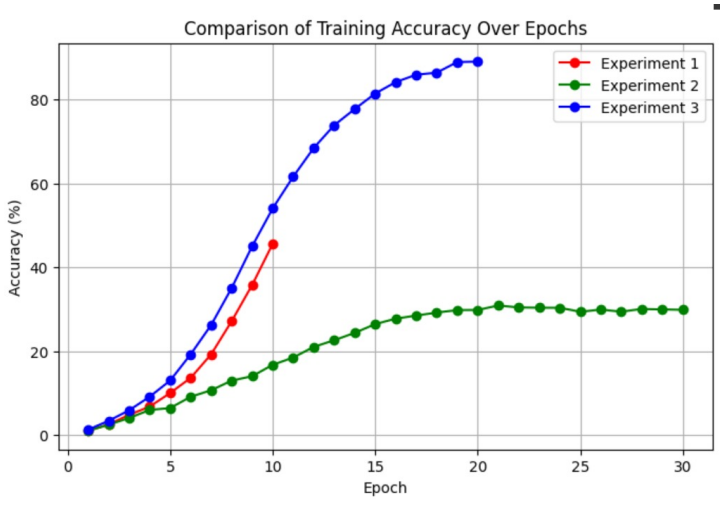

Figure 11: Comparison of Training Accuracy Over Epochs across End-to-End CNN experiments.

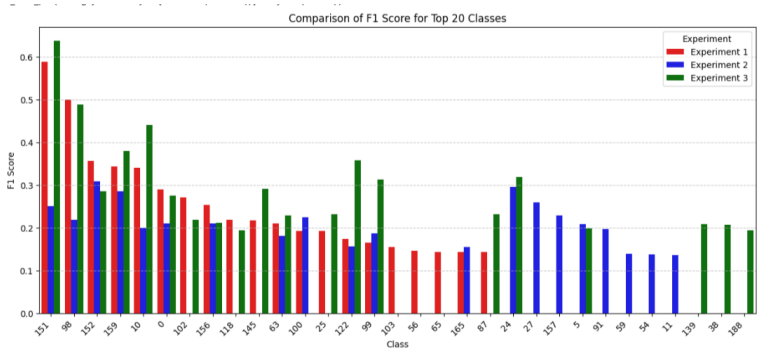

Figure 12: Comparison of F1 Score for Top 20 Classes across End-to-End CNN experiments.

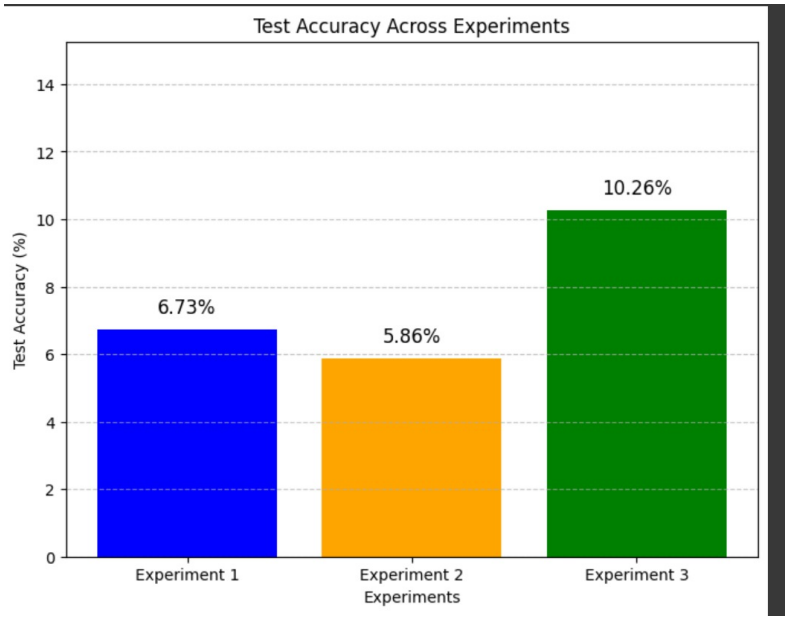

Figure 13: Comparison of Training vs Validation Loss across End-to-End CNN experiments.

4.2 Conclusion for End-to-End CNN

The final optimized CNN model with data augmentation achieved the best performance with **10.26%** test accuracy. However, results indicate that deep architectures such as EfficientNet-B7 (from previous configurations) perform significantly better with transfer learning and embeddings.

Lessons learned:

- **Regularization methods** such as dropout and batch normalization are crucial to mitigate overfitting.
- **Deeper architectures** improve feature extraction but require careful tuning of hyperparameters.
- **Augmentations** enhance model robustness but come at a higher computational cost.

4.3 Architectural Insights and Training Decisions

In addition to evaluating model performance, it is essential to analyze the impact of different architectural choices and training strategies on the final results.

4.3.1 Effect of Layers and Weight Initialization

Fully connected layers play a crucial role in transforming extracted features into final predictions. We observed that increasing the number of fully connected layers without proper regularization led to overfitting. Additionally, embedding layers significantly impacted feature representation quality, particularly in the Image Retrieval

experiments. Pretrained weights in transfer learning provided a strong foundation, but fine-tuning deeper layers proved necessary for improved performance.

4.3.2 Impact of Data Preprocessing and Image Resizing

Choosing the correct input image size is vital, as it affects both computational cost and model performance. We experimented with resizing images to different dimensions (e.g., 224×224, 128×128), observing that reducing image size too much resulted in loss of critical details. This impacted classification accuracy, especially for visually similar car models.

4.3.3 Regularization Techniques and Optimization Strategies

To mitigate overfitting, we applied several regularization techniques:

- **Dropout:** Prevented reliance on specific neurons and improved generalization.
- **Batch Normalization:** Stabilized training and enabled higher learning rates.
- **L2 Regularization:** Reduced weight magnitudes, preventing excessive complexity.
- **Early Stopping:** Stopped training when validation loss stopped improving, preventing unnecessary overfitting.

4.3.4 Challenges and Adjustments During Training

Throughout the training process, we encountered issues such as unstable validation loss and slow convergence. Adjustments were made by tuning the learning rate, experimenting with different optimizers (Adam vs. SGD), and scaling data augmentation intensity. The decision to fine-tune deep architectures was motivated by the observation that freezing early layers limited feature extraction potential.

4.3.5 Lessons Learned

Understanding how each architectural component contributes to overall performance was key. Transfer learning proved highly effective, but careful tuning of hyperparameters was required. Augmentation techniques, while beneficial, increased computational cost. Future work could explore automated hyperparameter tuning or advanced architectures such as transformers to further enhance results.

5 Best Performing Model and Test Environment

The best-performing model and its corresponding test environment were successfully identified for each configuration. The final optimized model was trained, and its learned weights were saved. Additionally, the test environment was defined and adjusted accordingly, ensuring a robust evaluation process. The model was also tested for its ability to connect to a given car image, demonstrating its effectiveness in classification.

Figure 14: Test environment setup for Notebook 1 (Transfer Learning experiments).

Figure 15: Test environment setup for Notebook 2 (Image Retrieval experiments).

Figure 16: Test environment setup for Notebook 3 (End-to-End CNN experiments).

References

[1] A. Krizhevsky et al., "Imagenet classification with deep convolutional neural networks," *NeurIPS*, vol. 25, pp. 1097-1105, 2012.

[2] K. He et al., "Deep residual learning for image recognition," *CVPR*, pp. 770-778, 2016.

6 Notebooks and Additional Resources

For further details and full experiment implementations, refer to the following notebooks:

- **NOTEBOOK1**
- **NOTEBOOK2**
- **NOTEBOOK3**