Basics Of Deep Learning - Final Project

Ofir Almog (ID. 207918731), Adir Edri (ID. 206991762)

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

1 Introduction

This project tackles the challenge of fine-grained car classification and image retrieval using deep learning. The dataset includes 16,185 labeled images across 196 car categories, where visual differences between classes are often subtle. Our goal was to design models that can accurately classify these images and retrieve visually similar car images given a query example.
We implemented three main configurations:

1. Transfer Learning with a pre-trained ResNet-50
2. Image Retrieval using embedding vectors and cosine similarity
3. End-to-End CNN trained from scratch

Each configuration was tested through a series of experiments, starting from basic setups and progressing to models with data augmentation and regularization. We evaluated performance using classification metrics (accuracy, F1 score, loss) and retrieval metrics (mAP) where appropriate.

1.1 Data

We used the cropped version of the Stanford Cars dataset, containing 16,185 labeled images across 196 car categories. The dataset was divided into 8,144 training images and 8,041 test images.
All images were resized to 224×224 and normalized using ImageNet statistics to ensure consistent input across models. These preprocessing steps were applied uniformly in all configurations.
Data augmentation techniques such as random cropping, flipping, and rotation were selectively applied in specific experiments to improve generalization. Detailed augmentation strategies are described in each relevant section.
This dataset is considered challenging due to high inter-class similarity, visual variability from lighting and backgrounds, and the relatively limited number of samples per class.

1.2 Problem

The task combines two goals: classifying each image into one of 196 visually similar categories, and retrieving images that are most visually similar to a given query image.
The main challenges were the subtle visual differences between car categories, the limited number of training images per class, and the variability in image conditions. We addressed these by experimenting with different configurations, applying data augmentation and regularization, and analyzing performance across multiple metrics.

2 Solution

We tackled the task of car classification and image retrieval using three distinct deep learning configurations, each with a different modeling strategy and network design:

1. **Transfer Learning:** A ResNet-50 (ImageNet) fine-tuned to classify 196 car categories.
2. **Image Retrieval:** Feature extraction using a frozen backbone, followed by KNN retrieval in embedding space via cosine similarity.
3. **End-to-End CNN:** A custom CNN trained from scratch, tailored specifically for fine-grained classification on our dataset.

Each configuration used a unique training pipeline and evaluation process. Results are detailed in the following sections.

2.1 Transfer Learning

2.1.1 General Approach

In this configuration, we applied Transfer Learning using a pre-trained ResNet-50 model originally trained on ImageNet. The goal was to leverage its learned features for our fine-grained car classification task with 196 categories. Instead of training from scratch, we replaced the final classification head with a new fully connected layer (2048→196) and fine-tuned the model on our dataset.
Throughout all experiments, the core backbone remained ResNet-50. In the early experiments, we froze the convolutional layers and only trained the final classification head. In later experiments, we progressively unfroze the backbone and added regularization such as dropout. These changes allowed us to evaluate how gradually unlocking layers and introducing augmentation impacted performance.

Optimizer	Learning Rate	Loss	Batch Size	Epochs	Scheduler
Adam	0.001	CrossEntropyLoss	32	10	StepLR (gamma=0.1 every 7)

Table 1: Shared training configuration across all experiments

All input images were resized to 224×224, and each model was trained to classify 196 car categories. The backbone used AdaptiveAvgPooling before the final fully connected layer. No additional learning rate scheduler was used beyond StepLR.
The following experiments analyze how performance evolved as we progressively introduced augmentation and regularization.

2.1.2 Experiment 1 – Base Model (Frozen Backbone)

We used a pre-trained ResNet-50 with all layers frozen except for the classification head. No data augmentation or dropout was applied. This experiment served as a baseline to evaluate the out-of-the-box transferability of ImageNet features.
Augmentation: None
Architecture: ResNet-50 with new FC layer (2048→196)
This experiment allowed us to isolate the quality of features extracted by the pre-trained backbone without any influence from training. Since only the classification head was updated, its performance reflects how transferable ImageNet representations are to fine-grained car categories. As expected, results were modest, highlighting the need for further tuning.

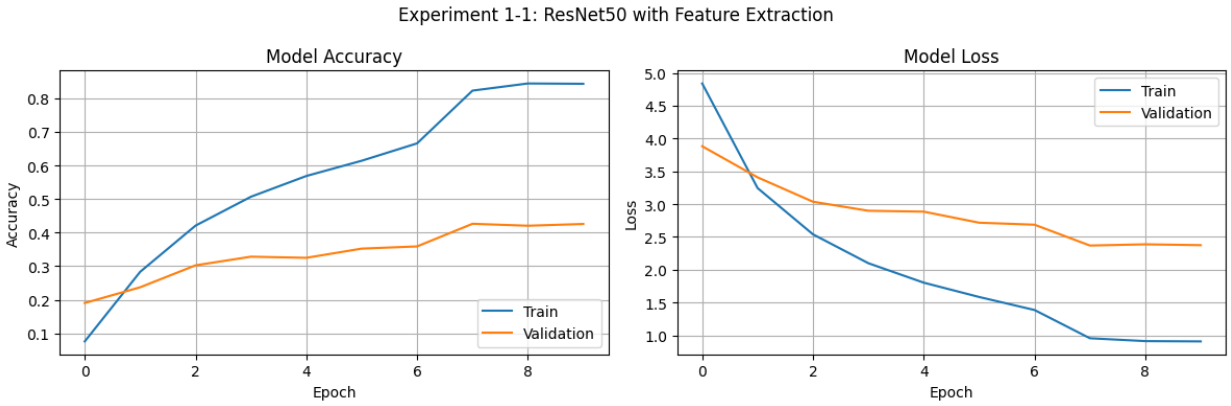

Figure 1: Accuracy and Loss curves – Experiment 1 (Frozen Backbone)

2.1.3 Experiment 2 – With Data Augmentation

We kept the same ResNet-50 architecture but introduced RandomCrop, HorizontalFlip, and Rotation(15°) to improve generalization. The backbone remained frozen throughout training.
These augmentations led to improved validation metrics, demonstrating that even a frozen network benefits from a more diverse data distribution.
Augmentation: RandomCrop, HorizontalFlip, Rotation(15°)
Architecture: Same as Experiment 1

2.1.4 Experiment 3 – With Augmentation and Dropout (Fine-Tuned)

In the final experiment, we fine-tuned the full ResNet-50 model by unfreezing all layers and added a Dropout($p = 0.5$) layer before the final classification head. We continued using the same augmentation strategy. This experiment achieved the best performance among all configurations.
Augmentation: Same as Experiment 2
Architecture: Same as before, with Dropout($p = 0.5$) before FC
Fine-tuning the entire network unlocked the full capacity of the model to learn domain-specific features. Combined with dropout and augmentation, this setup controlled overfitting and improved robustness. It demonstrates how gradual unfreezing and regularization can maximize transfer learning performance on limited data.

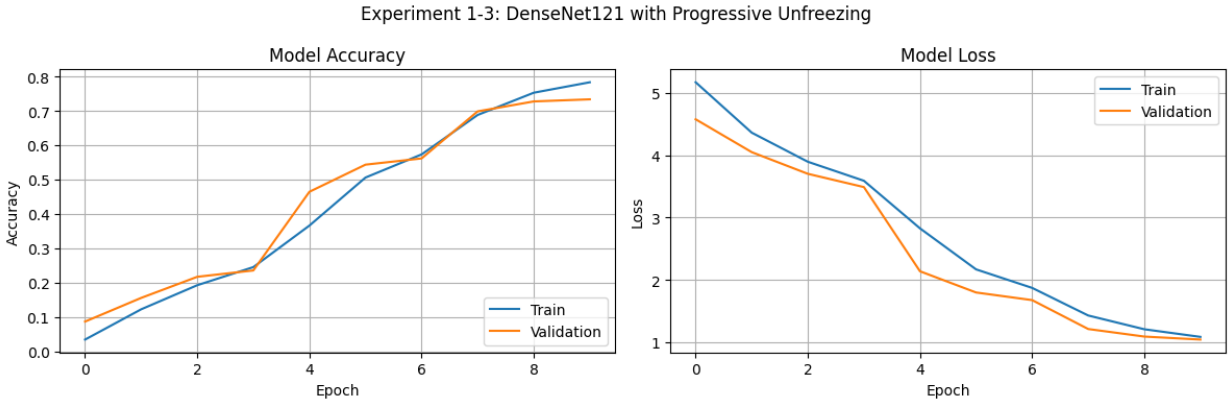

Figure 2: Accuracy and Loss curves – Experiment 3 (Fine-Tuned + Aug + Dropout)

2.1.5 Comparison and Best Model Selection

We compared all experiments using consistent evaluation metrics to determine the most effective configuration:

#	Model	Loss	F1 Score	Recall	Precision	Accuracy
1	ResNet50 – Frozen	2.3724	41.96%	42.37%	42.91%	42.37%
2	ResNet50 – Fine-Tuned	**0.9012**	**73.63%**	**73.78%**	**78.18%**	**73.78%**
3	DenseNet121 – Progressive	1.0800	71.70%	71.69%	75.43%	71.69%

Table 2: Comparison of evaluation metrics for Transfer Learning experiments

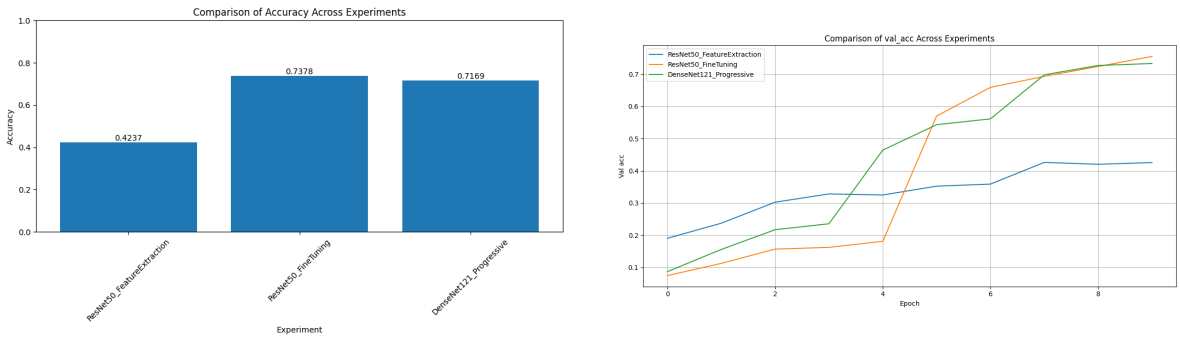

Figure 3: Validation Accuracy and Loss across Transfer Learning experiments

Best Model Summary:
The best-performing model was **ResNet50 – Fine-Tuned** (Experiment 3). Unfreezing the entire network and combining it with dropout and augmentation yielded the highest accuracy and F1 score across all configurations.
This experiment confirmed that with proper regularization and data enrichment, transfer learning can be highly effective even on fine-grained, multi-class tasks with limited data.

2.2 Image Retrieval

2.2.1 General Approach

In this configuration, we shifted from classification to visual similarity retrieval. Instead of directly predicting the class, we embedded the images into a feature space and retrieved their nearest neighbors based on similarity.
All experiments relied on a deep convolutional backbone to extract embeddings. We then applied a K-Nearest Neighbors (KNN) model to retrieve visually similar results. In the third experiment, we used cosine similarity, while in the others we used Euclidean distance.
The key differences between experiments include backbone type (ResNet50 or DenseNet121), training strategy (frozen vs. fine-tuned), and whether progressive unfreezing was applied. All experiments were trained for 10 epochs using consistent settings. The table below summarizes the shared and differing elements across the experiments:

#	Experiment	Backbone	Training Strategy	Feature Dim.
1	ResNet50 + KNN (k=3)	ResNet-50	Fine-tuned	2048
2	ResNet50 + KNN (k=10)	ResNet-50	Fine-tuned	2048
3	DenseNet121 + Cosine (k=7)	DenseNet121	Progressive unfreezing	1024

Table 3: Training and embedding setup for all experiments in the Image Retrieval configuration

2.2.2 Experiment 1 – ResNet50 + KNN (k=3)

In the first experiment, we fine-tuned a ResNet50 backbone on the classification task and used it to extract embedding vectors for all images. These embeddings were then fed into a KNN classifier with $k = 3$ and Euclidean distance. This experiment helped us evaluate how well the learned features capture visual similarity after transfer learning.
We intentionally kept k small to emphasize more localized neighborhoods in feature space, which suits fine-grained tasks. The results provided a strong initial baseline for retrieval performance.

2.2.3 Experiment 2 – ResNet50 + KNN (k=10)

In this setup, we reused the same fine-tuned ResNet50, but increased k to 10 to allow more diverse neighbor voting in ambiguous samples, improving generalization.
This adjustment led to performance improvements across most metrics. The wider neighborhood contributed to more robust retrieval when visual overlap existed between classes.

2.2.4 Experiment 3 – DenseNet121 + Cosine Similarity (k=7)

This experiment explored a different backbone, DenseNet121, which was trained using progressive unfreezing. We used cosine similarity instead of Euclidean distance, as it focuses on the direction of embeddings, which is more robust in normalized high-dimensional spaces.
While this configuration didn't outperform ResNet50 with $k = 10$, it achieved strong precision and the fastest inference time, making it a viable option for latency-sensitive applications.

2.2.5 Comparison and Best Model Selection

To choose the best configuration, we compared all experiments across multiple metrics:

#	Model	F1 Score	Recall	Accuracy	Precision	Fit Time	Predict Time
1	ResNet50 – KNN (k=3)	75.23%	74.93%	74.93%	77.38%	0.0092s	1.1998s
2	ResNet50 – KNN (k=10)	**76.77%**	**76.77%**	**76.77%**	**78.81%**	0.0079s	1.3873s
3	DenseNet121 – Cosine (k=7)	73.49%	73.41%	73.41%	75.33%	**0.0047s**	**1.0178s**

Table 4: Comparison of performance and runtime metrics across experiments

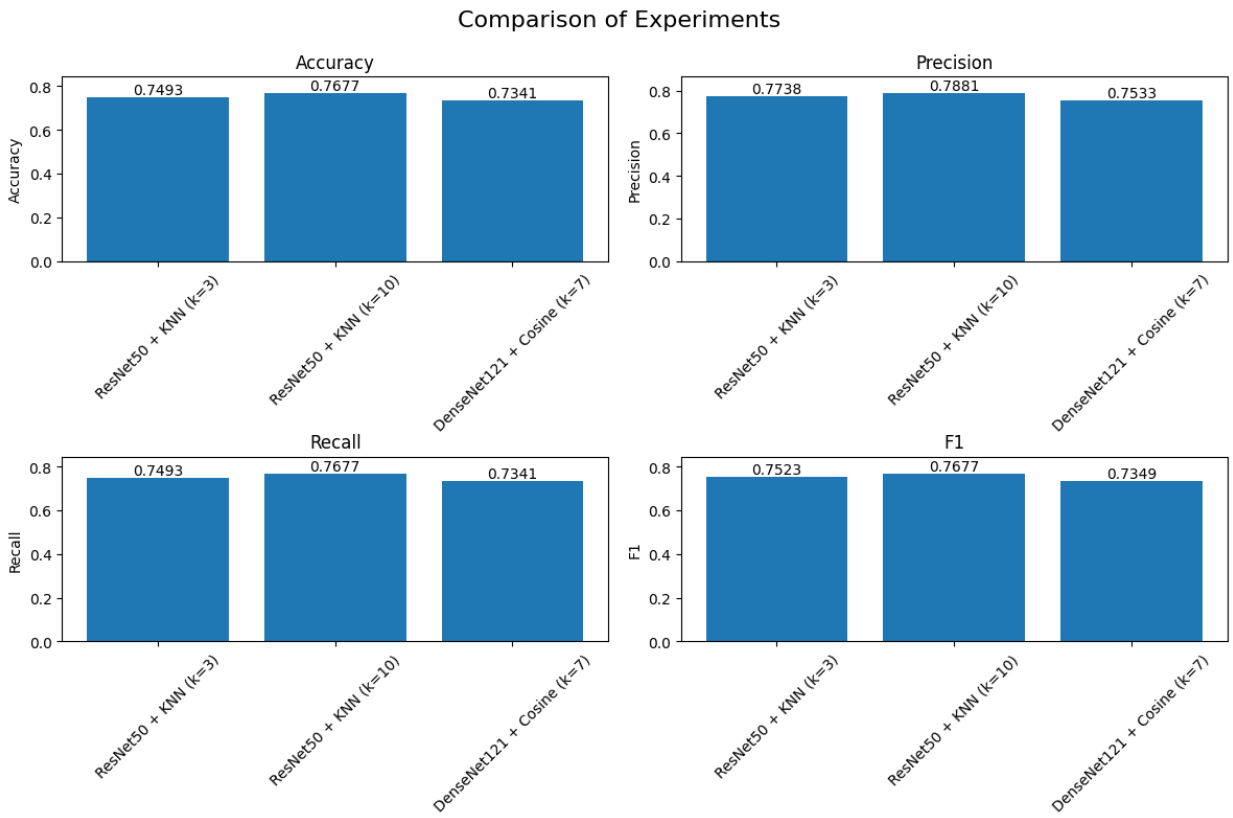

Figure 4: Metric comparison across experiments

Best Model Summary:
The best-performing model was **ResNet50 + KNN (k=10)** (Experiment 2). It achieved the highest accuracy and F1 score among all retrieval configurations, while maintaining low computational cost and a stable inference process.
We believe that the combination of fine-tuned ResNet50 embeddings with a wider k in the KNN search allowed the model to better generalize and capture inter-class similarities.
Although **DenseNet121 + Cosine (k=7)** (Experiment 3) had the lowest prediction time and respectable precision, it slightly underperformed in overall accuracy, making it more suitable in time-sensitive scenarios.
This model was deployed for real-time visual search, enabling users to retrieve similar car models based on image input.

2.3 End-to-End CNN

2.3.1 General Approach

In this configuration, we developed and trained a convolutional neural network (CNN) from scratch, without using any pre-trained weights. Unlike the previous configurations, we had full architectural control and the responsibility to learn all feature representations directly from the Cars-196 dataset. Designing a model from the ground up required careful tuning of the architecture and training pipeline, while addressing overfitting risks due to limited data and high inter-class similarity. We focused on gradually improving performance by adding depth, normalization, dropout, and data augmentation. All experiments in this configuration used the same core hyperparameters, shown below:

Loss Function	Optimizer	Learning Rate	Batch Size	Activation	Weight Init.
CrossEntropyLoss	Adam	0.001	64	ReLU	He Initialization

Table 5: Hyperparameter configuration used across all experiments

We used MaxPooling as the pooling method, resized all images to 224×224, and trained the models to classify 196 categories. No learning rate scheduler was used.
The table below summarizes the architectural and training changes across the three experiments:

#	Model	Conv Blocks	BatchNorm	Dropout	Augmentation	Epochs
1	Basic CNN	3	No	No	No	10
2	Modified DirectCNN	4	Yes	Yes (0.5)	No	15
3	Advanced CNN + Aug	4	Yes	Yes (0.5)	Yes	40

Table 6: Architectural and training differences across experiments

Next, we present each experiment and analyze the contribution of its structural and training modifications to overall model performance.

2.3.2 Experiment 1 – Basic CNN from Scratch

We started with a simple CNN containing three convolutional blocks, each followed by ReLU and MaxPooling, and concluded with a fully connected classification head. We applied no normalization, dropout, or augmentation.
This minimal setup struggled to capture useful patterns, especially in a high-resolution, fine-grained classification task like ours. It served as a basic performance baseline.

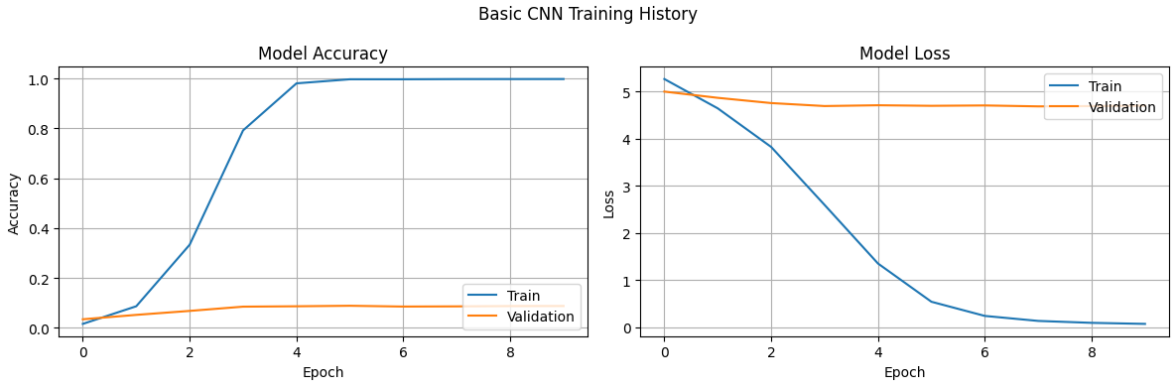

Figure 5: Training and Validation Curves – Experiment 1 (Basic CNN)

2.3.3 Experiment 2 – Modified DirectCNN with BatchNorm and Dropout

In this experiment, we enhanced the previous architecture by adding a fourth convolutional block and inserting Batch Normalization after each convolution. We also added Dropout ($p = 0.5$) before the final fully connected layer to reduce overfitting. The model was trained for 15 epochs.
While training was slightly more stable, the validation metrics remained weak. This setup highlighted that regularization alone wasn't enough, and the model still failed to generalize well to unseen examples.

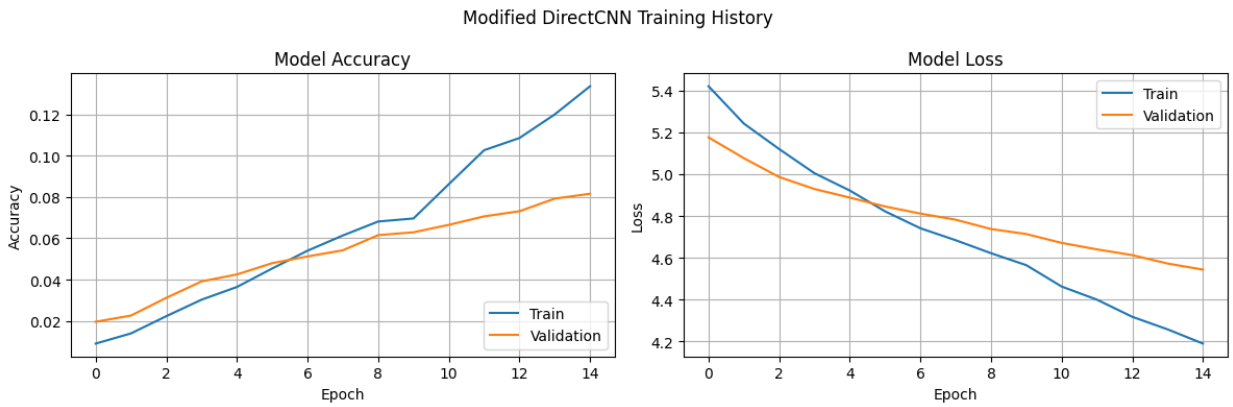

Figure 6: Training and Validation Curves – Experiment 2 (Modified DirectCNN)

2.3.4 Experiment 3 – Advanced CNN with Data Augmentation

In our final experiment, we kept the architecture from Experiment 2 and added several data augmentation techniques to the training pipeline, including RandomCrop, HorizontalFlip, and ColorJitter. We also extended training to 40 epochs.
This change resulted in a significant performance leap. The combination of regularization layers, deeper architecture, and rich augmentation allowed the model to learn more robust and generalizable features.
We believe the longer training duration contributed as well, giving the model more time to converge and leverage the augmented data distribution effectively. It's clear that this experiment marks the turning point in the success of training a CNN from scratch in our setting.

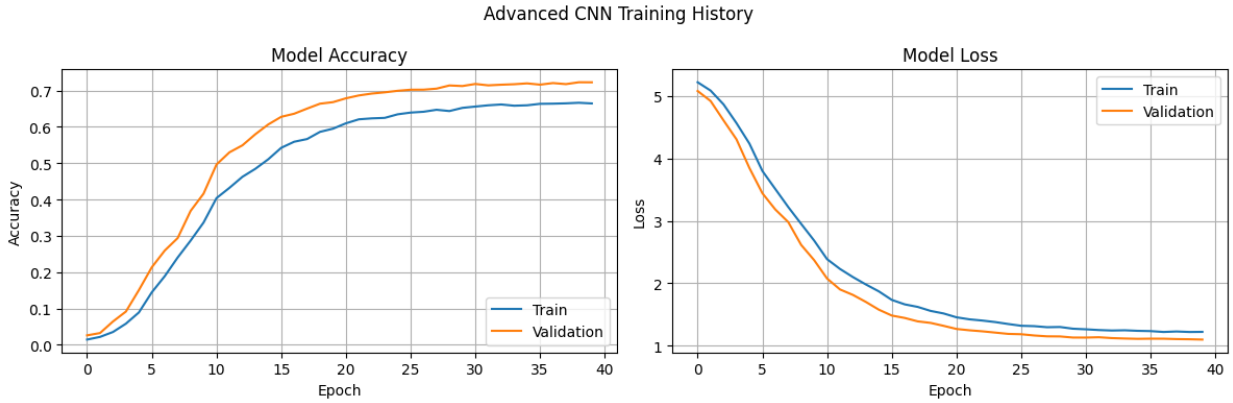

Figure 7: Training and Validation Curves – Experiment 3 (Advanced CNN + Augmentation)

2.3.5 Comparison and Best Model Selection

We compared all experiments using a consistent evaluation protocol. The metrics are summarized below:

#	Model	Loss	Accuracy	F1 Score	Recall	Precision
1	Basic CNN	4.6955	8.88%	8.52%	8.88%	9.80%
2	Modified DirectCNN	4.5436	8.16%	6.23%	8.16%	7.17%
3	Advanced CNN + Aug	**1.1026**	**72.26%**	**72.17%**	**72.26%**	**72.99%**

Table 7: Comparison of metrics for End-to-End CNN experiments

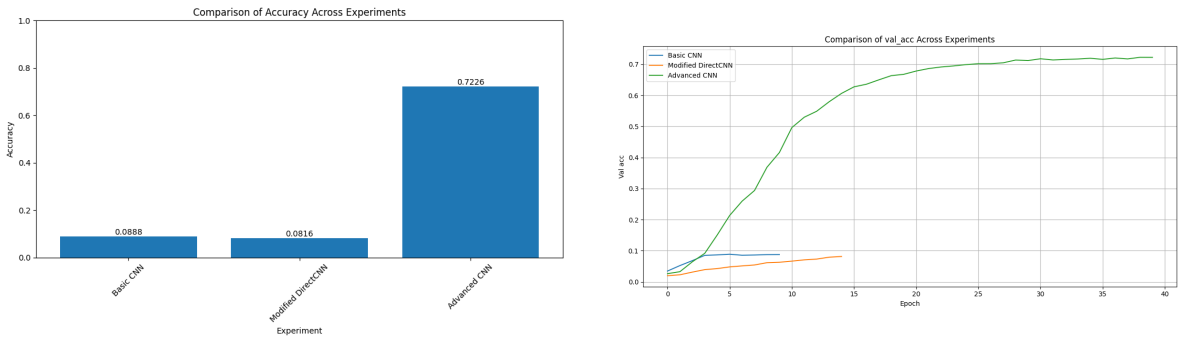

Figure 8: Validation Accuracy and F1 Score across all CNN experiments

Best Model Summary:
The best-performing model was **Advanced CNN + Augmentation** (Experiment 3). It dramatically outperformed the other setups, reaching a validation accuracy of 72.26% and F1 score of 72.17%.
This highlights the importance of using a rich data pipeline and well-structured architecture when training from scratch. The combination of deeper layers, normalization, dropout, and augmentation allowed the network to learn more robust and generalizable features.
We selected this model as our final choice for custom CNN classification and believe it could serve as a strong foundation for future ensemble approaches or further improvements.

2.4 Discussion

This project explored fine-grained car classification and image retrieval through three distinct deep learning strategies: Transfer Learning, Visual Similarity Retrieval, and End-to-End CNNs. Each approach brought unique strengths and challenges, helping us evaluate which strategy works best under different constraints.

Interestingly, while the Image Retrieval configuration (Conf 2) reached the highest **top-k accuracy** of 76.77%, its purpose was different: retrieving similar images rather than assigning a single class label. Therefore, when comparing all configurations under a pure classification lens, the most reliable and stable model was the one trained with **Transfer Learning** (Conf 1), achieving 73.78% accuracy with a significantly lower training cost and faster convergence.

The success of Transfer Learning highlights the power of leveraging pre-trained models, especially in data-scarce, fine-grained settings. By fine-tuning a ResNet-50 backbone trained on ImageNet, we benefited from strong general visual features, reduced overfitting, and faster training. The final model in this configuration outperformed all others in terms of classification metrics, including **F1 Score (73.63%)** and **Precision (78.18%)**, making it the most suitable for real-world deployment.

The End-to-End CNN (Conf 3) demonstrated how architecture and training strategies like BatchNorm, Dropout, and augmentation impact performance. Although it required extensive tuning and 40 epochs to converge, it eventually reached respectable accuracy (72.26%) and proved that with sufficient regularization and data augmentation, training from scratch is viable.

Overall, the experiments emphasized how choices in initialization, data processing, and model design significantly affect learning dynamics and final outcomes. Transfer Learning was ultimately the best balance between performance, effort, and scalability — especially for fine-grained classification tasks with limited data.

We believe this project deepened our practical understanding of deep learning workflows, model selection, and training techniques in real-world fine-grained image recognition tasks.

3 Code

Our implementation was done entirely in PyTorch and organized into three Google Colab notebooks, each corresponding to one of the configurations in the project.

Links to the Colab notebooks:

- **Part 1 -** Transfer Learning Notebook Link
- **Part 2 -** Image Retrieval Notebook Link
- **Part 3 -** End-to-End CNN Notebook Link