Basics Of Deep Learning - Final Project

Alon Vizniuk, AlonVizniuk@gmail.com

Submitted as final project report for Basics Of Deep Learning course, Colman, 2024

1 Introduction

This project explores car image classification using three different deep learning approaches: transfer learning , image retrieval with feature embeddings and KNN, and a custom CNN model. The goal is to compare these methods and evaluate their effectiveness on the Stanford Cars dataset, which consists of various car models.

1.1 Data

The project utilizes the Stanford Cars dataset, which contains 16,185 images of 196 different car models. Originally divided into 8,144 training examples and 8,041 testing examples, we modified the dataset to include a validation set. Each image is labeled according to its corresponding car model. The images vary in size and were captured under different conditions, adding complexity to the classification task

1.2 Problem

The problem is to classify car images into one of 196 categories based on the car model. It is a fine-grained classification task where subtle differences between similar models make it challenging. The images vary in angle, lighting, and background, requiring effective deep learning models for accurate classification.

2 Solution

In this section, we will present the various solutions we explored to solve the car image classification problem using deep learning. The main focus will be on three different approaches: transfer learning, image retrieval using feature embeddings and KNN, and a custom CNN model.

2.1 Transfer Learning

2.1.1 General Approach

We applied transfer learning using a pre-trained ResNet50 model, fine-tuning it for car image classification. This approach allows us to leverage the knowledge from a large dataset like ImageNet, reducing the need for extensive data and training time while adapting the model to our specific task.

Model Selection: We selected ResNet50 for its strong performance and widespread use in deep learning, particularly for visual tasks. Its deep architecture, with the use of residual connections, enables efficient training even in very deep networks. Additionally, the pre-trained weights on ImageNet provide a solid starting point, making it well-suited for fine-tuning on domain-specific datasets like ours.

Data Preprocessing: We organized the images into class-specific folders for compatibility with PyTorch's ImageFolder class. Additionally, we split the original test set into two equal parts to create a validation set, resulting in a roughly 50%-25%-25% distribution of the data between the training, validation, and test sets.

Augmentation: For data augmentation, we chose random horizontal flipping, rotation, and resizing for the training set to enhance image variety and help prevent overfitting. These augmentations simulate real-world variations in the data, making the model more robust. Resizing the images to 400x400 pixels ensures uniformity across the dataset. For the test set, we kept the transformations simpler with resizing and normalization to preserve consistency with the training set and avoid introducing any additional variations that could affect the model's performance on unseen data.

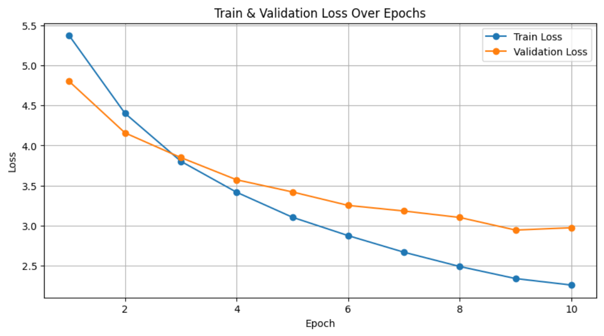
Fine-tuning: For fine-tuning, we began by freezing all layers of the ResNet50 model to retain the knowledge learned from ImageNet. Then, we unfreezed some of the layers to adapt the model to our specific dataset. Additionally, we replaced the final layer with a new one consisting of 196 neurons, corresponding to the number of car models in our dataset. This allowed the model to learn task-specific features while preserving the lower-level features from ImageNet.

2.1.2 Base-Model

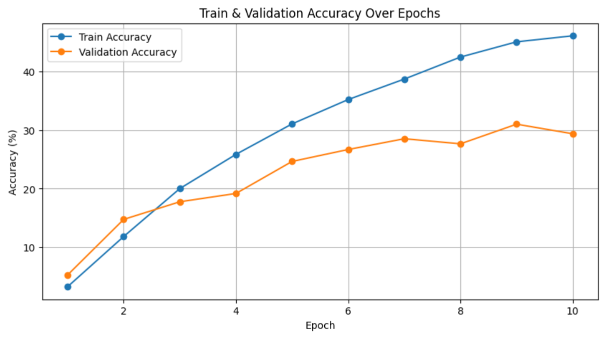
In the base model experiment, we unfreezed only the last fully connected layer and chose relatively standard hyperparameters, based on previous tasks where they proved effective. Below are the key hyperparameters and training setup:

- **Optimizer:** Adam (lr=0.001) for efficient fine-tuning.
- **Loss Function:** CrossEntropyLoss for multi-class classification.
- **Batch Size:** 32, balancing memory and stability.
- **Epochs:** Trained for 10 epochs.

Base Model Experiment Training Results:



Training and Validation Loss Over Epochs



Training and Validation Accuracy Over Epochs

The graphs show the training and validation loss and accuracy over 10 epochs. The training loss decreases, and accuracy increases, as expected. However, the validation metrics plateau after a few epochs, suggesting overfitting, with only minor improvements from epoch 5 onwards. The model finishes with a validation accuracy of 29.38% far from the 46.12% of the training set.

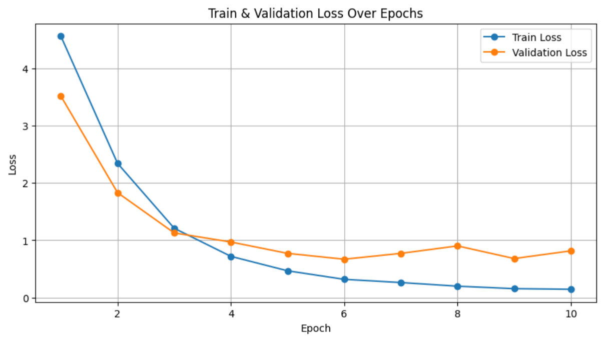
2.1.3 Experiment 1

In this experiment, we fine-tuned the last three layers of ResNet50, reduced the batch size to 16, and introduced a learning rate scheduler to enhance training stability and prevent overfitting.
Below are the key hyperparameters and training setup:

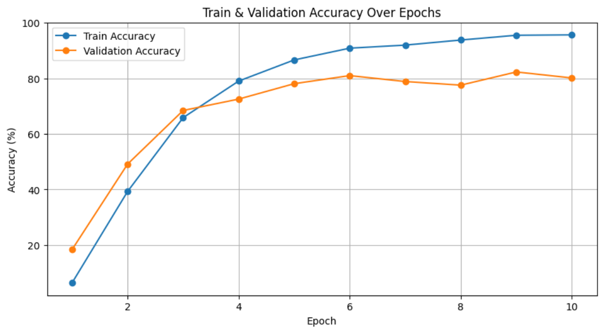
- **Optimizer:** Adam with a learning rate of 0.005 for faster convergence.

- **Loss Function:** CrossEntropyLoss for multi-class classification.
- **Batch Size:** 16, to allow more frequent updates and prevent overfitting.
- **Epochs:** 10, as in the base model.
- **Fine-Tuning Strategy:** Fine-tuned the last three layers of ResNet50, compared to only the final layer in the base model.
- **Learning Rate Scheduler:** Used ReduceLROnPlateau to reduce the learning rate by a factor of 0.1 if the validation performance doesn't improve for 3 epochs.

Experiment 1 Training Results:



Training and Validation Loss Over Epochs



Training and Validation Accuracy Over Epochs

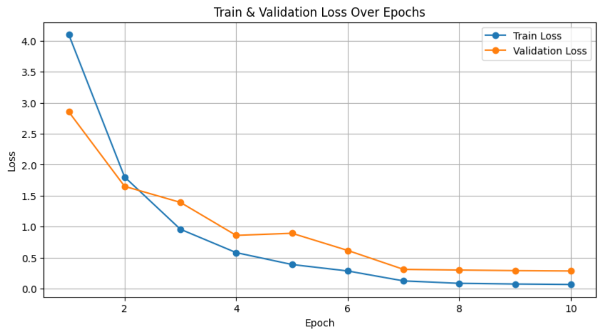
Experiment 1 shows steady improvement in both training and validation accuracy. Initial gains were significant, with validation accuracy rising rapidly in early epochs. Although validation loss fluctuated, the overall trend was positive, indicating effective learning. The learning rate scheduler helped balance overfitting while maintaining progress, enhancing the model's performance.

2.1.4 Experiment 2

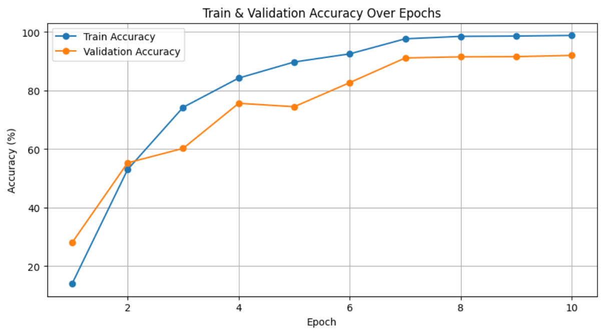
In Experiment 2, we fine-tuned all layers of ResNet50, returned the batch size to 32, and switched to the SGD optimizer with momentum for improved training stability. Additionally, we adjusted the learning rate scheduler's threshold to 0.9 for more adaptive learning rate adjustments. Below are the key hyperparameters and training setup:

- **Optimizer:** We chose SGD with momentum (lr=0.01) over Adam because it helps achieve smoother convergence, especially in fine-tuning pre-trained models. The momentum prevents the optimizer from getting stuck in local minima, allowing for more stable and faster convergence in deep models like ResNet50.
- **Loss Function:** CrossEntropyLoss for multi-class classification.
- **Batch Size:** 32, based on prior results indicating minimal impact on performance.
- **Epochs:** 10, same as the base before.
- **Fine-Tuning Strategy:** Fine-tuned all layers of ResNet50, unlike the previous experiments where most of the layers were frozen.
- **Learning Rate Scheduler:** Used ReduceLROnPlateau with a threshold of 0.9 to reduce the learning rate if the validation performance plateaus.

Experiment 2 Training Results:



Training and Validation Loss Over Epochs



Training and Validation Accuracy Over Epochs

In Experiment 2, the model demonstrated significant improvement in both training and validation performance over the course of 10 epochs. The training accuracy increased steadily, reaching 98.83% by the final epoch, while the validation accuracy peaked at 92.01%. The validation loss decreased sharply in the initial epochs and stabilized, indicating that the model effectively learned to generalize from the training data. Despite the drop in validation accuracy towards the later epochs, the overall trend shows good convergence and stability, with a slight reduction in overfitting compared to Experiment 1.

2.1.5 Model Comparisons

Model	**Test Loss**	**Test Accuracy**	**Precision**	**Recall**	**F1-Score**
Base Model	3.0628	28.10%	0.5675	0.2810	0.2773
Experiment 1	0.7619	79.88%	0.8360	0.7988	0.7992
Experiment 2	0.2930	92.14%	0.9248	0.9214	0.9212

Table 1: Precision, Recall, and F1-Score are reported as weighted averages.

2.1.6 Conclusions

In the Transfer Learning part, we fine-tuned the ResNet50 model for car classification. Experiment 2, which unfreezed all layers and used SGD with momentum, showed the best results, demonstrating that fine-tuning and careful hyperparameter selection greatly enhanced the model's performance.

2.2 Image Retrieval

2.2.1 General Approach

For the image retrieval task, we built upon the best-performing model from the transfer-learning part (Experiment 2), as it was already fine-tuned to our car classification dataset. Using this model for image retrieval provides an advantage over using a standard ResNet50 model, as it has learned domain-specific features that improve the quality of the feature embeddings, making retrieval more accurate. The following key data processing changes were made:

- **Image Resolution:** Images were resized to 400x400 pixels to match the input size used during training in Experiment 2, ensuring consistency during feature extraction.
- **Normalization:** We applied normalization with a mean and standard deviation of (0.5, 0.5, 0.5) for each color channel, consistent with the preprocessing applied during fine-tuning in Experiment 2.

Embedding Layer and Model Modifications:

- **Model Modifications:** We removed the final fully connected layer of the Exp2 model, leaving only the convolutional layers to act as the feature extractor. This allows the model to generate embeddings for input images.
- **Feature Extraction:** The ResNet50 model extracts 2048-dimensional embeddings, which are compact numerical representations of images. These embeddings capture essential features, enabling efficient comparison and retrieval of similar images.
- **KNN Retrieval:** Using the extracted embeddings from the training dataset, we performed image retrieval for the test set by applying the k-nearest neighbors (KNN) algorithm. We experimented with several distance metrics and different values of K to optimize the retrieval. After finding the closest neighbors, we applied majority voting to predict the class of each test image based on the most common class among its nearest neighbors.

2.2.2 Base-Model

In this experiment, we trained a KNN model with $k = 3$ using the Manhattan distance metric.

- **Distance Metric:** We chose the Manhattan distance metric as it is computationally efficient and runs faster compared to other distance metrics like Euclidean, especially in high-dimensional spaces.
- **Choice of k:** The value of k was chosen as 3, which is a small odd number to simplify the majority voting process, ensuring a clear classification when the decision is between two classes.

Base-Model Results:

Metric	**Value**
Accuracy	56.50%
Precision	72.36%
Recall	56.43%
F1-score	57.05%

Table 2: Base-Model Results with $k = 3$ and Manhattan distance metric

Although the results of the base model were not so bad, there is significant room for improvement.

2.2.3 Experiment 1

In side experiments, we tested different values of k and distance metrics for KNN, but these changes did not improve retrieval performance. After researching, we found that applying L2 normalization to embeddings is a common technique to enhance performance, so we decided to apply it and see if it would improve the results.

- **L2 Normalization:** The embeddings were normalized using L2 normalization, scaling each vector so that its magnitude becomes 1 while maintaining its direction. For example, for the vector $\mathbf{v} = [3, 4]$, the L2 norm is 5, and the normalized vector becomes $[0.6, 0.8]$.
- **KNN Training:** The model was trained with same k and distance metric just like the previous experiment, but with normalized embeddings.

The only change from the base model to this experiment was applying L2 normalization to the embeddings, making this a fair comparison where the only factor influencing the improvement is the normalization.

Experiment 1 (Normalized Embeddings) Results:
The results of Experiment 1 show a significant improvement over the base model, with accuracy, precision, recall, and F1-score all reaching above 89%. This demonstrates that applying L2 normalization to the embeddings enhanced the retrieval performance, highlighting the effectiveness of this technique.

Metric	Value
Accuracy	89.42%
Precision	89.79%
Recall	89.37%
F1-score	89.35%

Table 3: Experiment 1 Results with $k = 3$ and Manhattan distance metric

2.2.4 Experiment 2

After applying L2 normalization in the previous experiment, we observed significant improvements. In this experiment, we focused on selecting the optimal distance metric and tuning the value for k.

- **KNN Training:** The KNN model was trained with $k = 13$ using the Manhattan distance metric on the normalized embeddings. We selected $k = 13$ as it provided the best performance across metrics, and the Manhattan distance was chosen for its simplicity and faster computation, consistently outperforming Euclidean and Cosine distances.
- **Evaluation:** Test images were classified using simple majority voting, as in previous experiments.

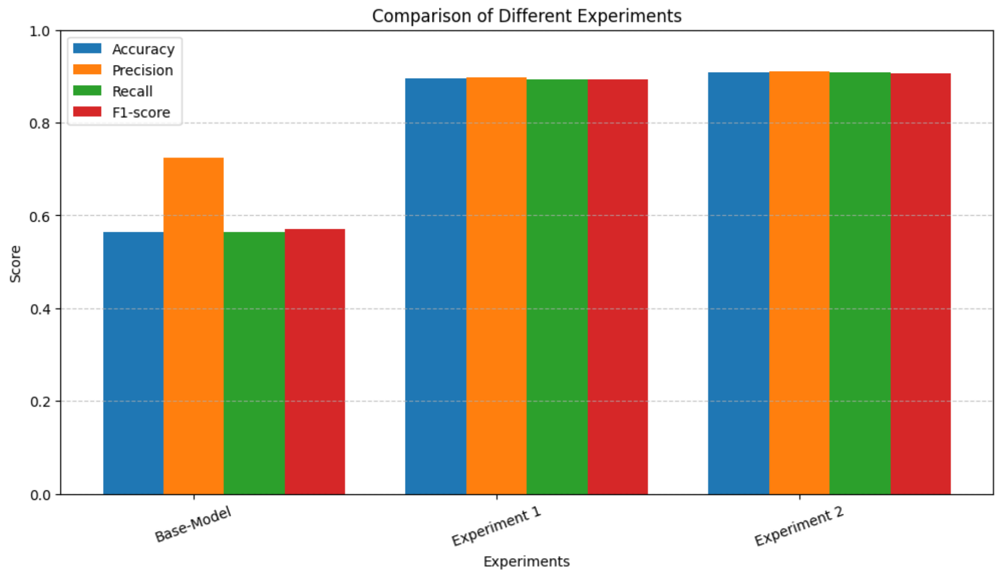
Experiment 2 (Normalized Embeddings) Results:

Metric	Value
Accuracy	90.80%
Precision	91.11%
Recall	90.74%
F1-score	90.71%

Table 4: Experiment 2 Results with $k = 13$ and Manhattan distance metric

The results of this experiment represent the best we achieved after testing various combinations. While the improvement over the previous experiment is modest, a difference of over 1% is still noteworthy.

2.2.5 Model Comparisons



Metric	Base-Model	Experiment 1	Experiment 2
Accuracy	0.5650	0.8942	0.9080
Precision	0.7236	0.8979	0.9111
Recall	0.5643	0.8937	0.9074
F1-score	0.5705	0.8935	0.9071

Table 5: Comparison of Metrics Across All Experiments

2.2.6 Conclusions

In the Image Retrieval section, we extracted embeddings from the fine-tuned ResNet50 model and applied L2 normalization, which improved performance. Experiment 2, with $k = 13$ and Manhattan distance, achieved the best results.

2.3 End-to-End CNN

2.3.1 General Approach

In the End-to-End CNN section, we aimed to train a custom CNN model from scratch for car image classification problem.

- **Data Preparation:** We organized the dataset by class and applied different augmentations compared to the transfer learning section. For training, we used random horizontal flipping, random rotation, and **ColorJitter** for brightness, contrast, and saturation variations to improve generalization. We resized images to 256x256, followed by random cropped resizing to 224x224 for faster training. The test set was resized to 256x256 with center cropping to 224x224 for consistency.
- **Model Training:** The CNN model used convolutional layers to extract low-level features, which were then combined to capture more complex patterns. Fully connected layers transformed these features into final classification predictions.
- **Data Loaders:** We split the dataset (70%-15%-15%) for training, validation, and testing, applying appropriate transformations. The goal was to increase the training set size to help the model learn more diverse cases and improve generalization.

2.3.2 Base Model & Experiment 1

In this section, we define the architecture of our custom CNN model and describe the training process for both the Base Model and Experiment 1.

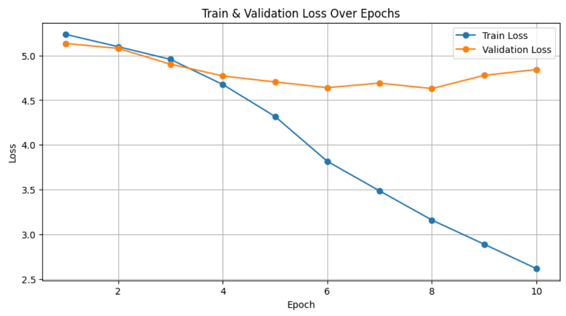
Custom CNN Architecture: The CustomCNN model consists of three convolutional layers, followed by ReLU activations and MaxPooling to extract features and reduce spatial size. The output is then flattened and passed through two fully connected layers for classification. The structure is as follows:

- **Convolutional Layers:**
  - Conv2D($3 \rightarrow 64$) $\rightarrow$ ReLU $\rightarrow$ MaxPool(2,2)
  - Conv2D($64 \rightarrow 128$) $\rightarrow$ ReLU $\rightarrow$ MaxPool(2,2)
  - Conv2D($128 \rightarrow 256$) $\rightarrow$ ReLU $\rightarrow$ MaxPool(2,2)
- **Fully Connected Layers:**
  - FC1 (512 neurons) $\rightarrow$ ReLU
  - FC2 (196)

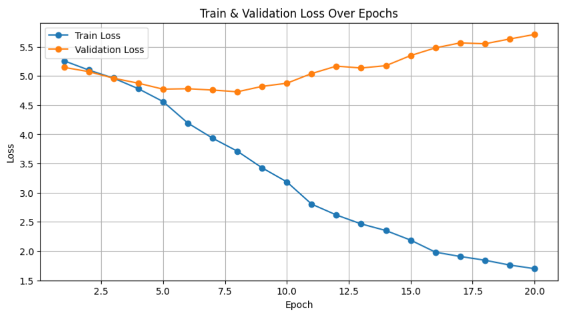
Base-Model Training: For the base model, we used CrossEntropyLoss as the loss function and Adam optimizer with a learning rate of 0.0003. The StepLR scheduler reduced the learning rate by half every 5 epochs. The model was trained for 10 epochs with a batch size of 32.

Experiment 1 Training: In Experiment 1, we used CrossEntropyLoss again and Adam optimizer with a raised learning rate of 0.0005. The StepLR scheduler halved the learning rate every 5 epochs. We train the model for 20 epochs with a batch size of 64, compared to 10 epochs and batch size of 32 as before.

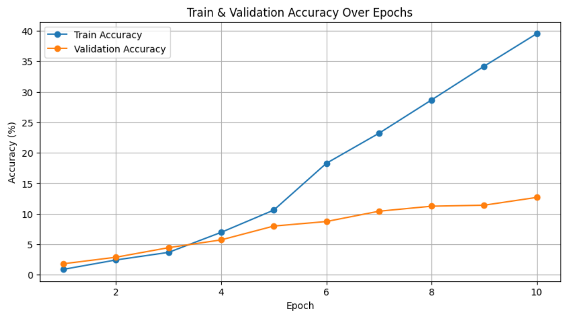
Base-Model & Experiment 1 Results:



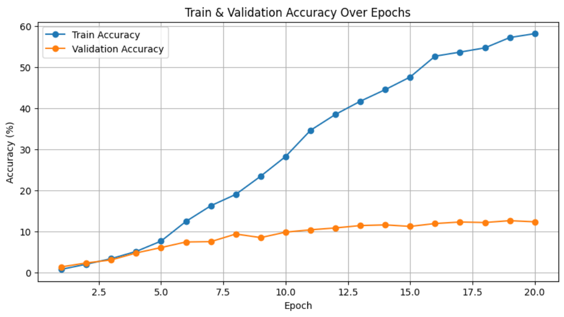
Base-Model Training and Validation Loss



Experiment 1 Training and Validation Loss



Base-Model Training and Validation Accuracy



Experiment 1 Training and Validation Accuracy

Both experiments show significant signs of overfitting, as evidenced by the graphs. In both the Base-Model and Experiment 1, the training accuracy improves significantly, while the validation accuracy remains relatively low. More importantly, in both experiments, the Validation Loss starts to increase after a certain number of epochs, indicating that the model is not generalizing well. These results highlight the need for further adjustments to prevent overfitting and improve generalization performance.

2.3.3 Experiment 2

After poor results from previous experiments and several hyperparameter combinations, we concluded that modifying the network architecture by adding layers and increasing its complexity was necessary for better performance.

New Custom CNN Architecture: In Experiment 2, we enhanced the model by introducing several improvements to stabilize training and improve generalization. These changes include:

- **Batch Normalization:** Batch Normalization was added after each convolutional layer to normalize the activations and gradients during training. This helps in reducing internal covariate shift, stabilizing learning, and speeding up convergence.

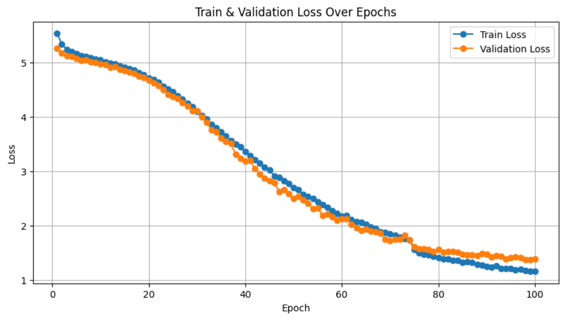
- **Dropout:** Dropout layers were added to prevent overfitting. We used a spatial dropout of 0.1 after each convolutional layer and a 0.5 dropout rate in the fully connected layers. Dropout randomly sets a fraction of input units to zero at each update during training, which helps the model generalize better by reducing reliance on specific neurons.
- **Increased Depth:** Another convolutional layer (Conv2D(128 → 256)) was added to increase the model's capacity to learn more complex features. This helps the model better capture intricate patterns in the data.
- **Global Average Pooling (GAP):** Instead of flattening the feature maps, we applied Global Average Pooling. GAP reduces the spatial dimensions to a single value for each feature map, providing a more compact and generalized representation of the features and reducing the risk of overfitting.

The new architecture is as follows:

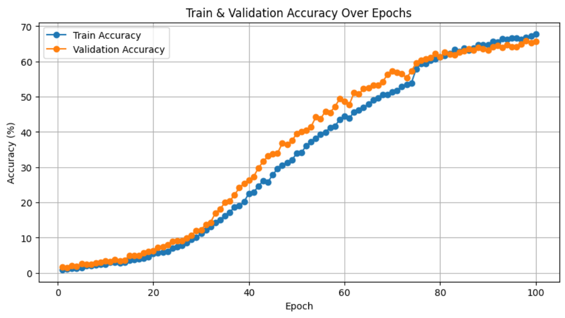
- **Convolutional Layers:**
  - Conv2D(3 → 32) → BatchNorm → ReLU → MaxPool(2,2) → Dropout (0.1)
  - Conv2D(32 → 64) → BatchNorm → ReLU → MaxPool(2,2) → Dropout (0.1)
  - Conv2D(64 → 128) → BatchNorm → ReLU → MaxPool(2,2) → Dropout (0.1)
  - Conv2D(128 → 256) → BatchNorm → ReLU → MaxPool(2,2)
- **Fully Connected Layers:**
  - Global Average Pooling (GAP)
  - FC1 (512 neurons) → BatchNorm → ReLU → Dropout (0.5)
  - FC2 (196)

Experiment 2 Training: In Experiment 2, we used CrossEntropyLoss as the loss function and the AdamW optimizer with a learning rate of 0.001 and weight decay of 0.01 for better weight regularization. AdamW is chosen because it decouples weight decay from the learning rate, allowing for better generalization. The ReduceLROnPlateau scheduler was employed to reduce the learning rate when the validation loss plateaued. This ensures the model can converge more efficiently. After running several epochs and observing that the model was learning slowly but not suffering from overfitting we chose to extend the training to 100 epochs to allow the model more time to improve.

Experiment 2 Results:



Experiment 2 Training & Validation Loss



Experiment 2 Training & Validation Accuracy

The graphs show consistent improvement in both training and validation loss, with minimal gap between them, indicating good generalization. Training accuracy reached 67%, while validation accuracy improved to 65.64%, suggesting the model is learning effectively without overfitting.

An interesting observation is that towards the end, the validation accuracy starts to plateau while training accuracy continues to rise. While additional epochs may improve results, we wouldn't expect the strong correlation between the two sets to continue, hinting at potential overfitting with further training.

2.3.4 Model Comparisons

Metric	**Base Model**	**Experiment 1**	**Experiment 2**
Test Loss	4.7953	5.5960	1.3460
Test Accuracy	12.76%	14.33%	65.54%
Precision (Macro Avg.)	0.1448	0.1423	0.6621
Recall (Macro Avg.)	0.1310	0.1426	0.6550
F1-Score (Macro Avg.)	0.1222	0.1363	0.6420

Table 6: Comparison of Performance Metrics Across the Three Experiments

2.3.5 Conclusions

We improved the custom CNN model across multiple experiments. The first architecture showed low performance. In Experiment 2, with added batch normalization, dropout, and increased depth, the model's performance significantly improved, achieving 65.64% validation accuracy. The results suggest effective learning without overfitting, but further training might risk overfitting. Overall, the changes led to better generalization and model performance. As a future project, it would be interesting to train this model for more epochs to explore its performance further.

3 Discussion

This project focused on three main areas using the Stanford Cars dataset: transfer learning, image retrieval, and custom CNN development.

- **Transfer Learning:** We used the ResNet50 model for feature extraction, improving classification performance with less data.
- **Image Retrieval:** We implemented a retrieval system using KNN based on the normalized embeddings from our pre-trained ResNet50, enabling efficient image search.
- **Custom CNN:** We improved the CNN architecture with batch normalization, dropout, and increased depth, leading to better generalization and a 65.64% validation accuracy.

In summary, the project demonstrated the effective use of transfer learning, image retrieval, and custom CNN improvements. The integration of pre-trained models for feature extraction and the optimization of the CNN architecture resulted in enhanced model performance and efficient image retrieval, paving the way for further research in these areas.