Assignment 1 – Deep Learning Workshop

omri arie, Liel Layney

1. data analysis:
   a. We choose to work with Stanford cars dataset.
   The dataset has 16,195 photos in total with 196 unique classes
   - training folder: 8154
   - test folder: 8041

   b.
   Each sample contains:
   - Image data (RGB channels, 3 dimensions)
   - Bounding box coordinates for the car location
   - Class label (out of 196 unique classes)

   Data preprocessing needed:
   - Resize images to consistent dimensions
   - Extract car regions using provided bounding boxes
   - Normalize pixel values

   Data augmentation possibilities:
   - Horizontal flips (valid since cars can be viewed from either side)
   - Small rotations (within reason to maintain realism)
   - Color jittering (brightness/contrast adjustments)

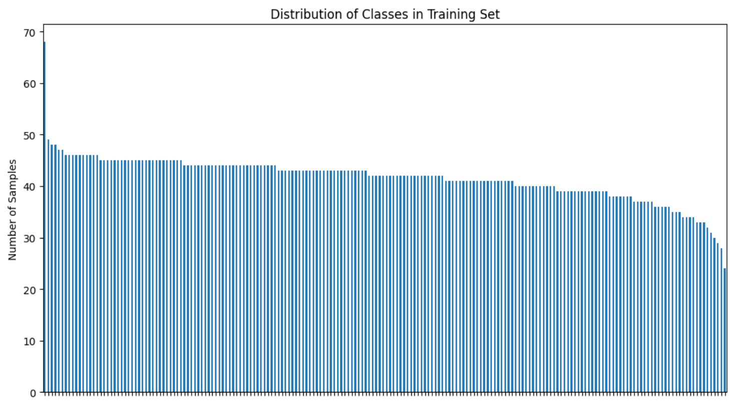
   c. Based on the class distribution statistics and visualizations, the dataset appears to be relatively balanced:

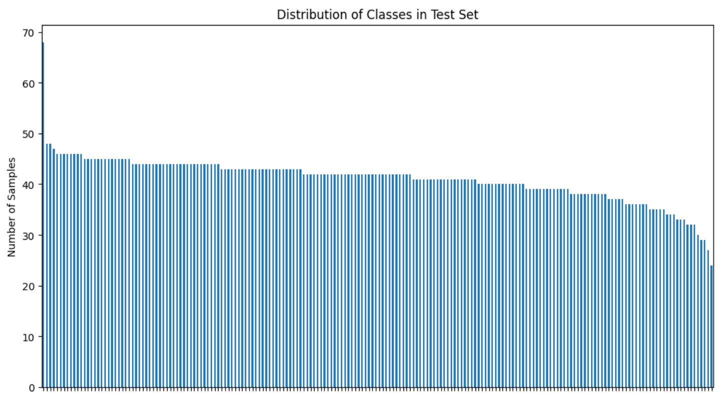
   Training Set:

   - Mean samples per class: 41.55
   - Min samples per class: 24
   - Max samples per class: 68

   Test Set:

   - Mean samples per class: 41.03
   - Min samples per class: 24
   - Max samples per class: 68





The distribution plots show that both training and test sets have a fairly uniform distribution across classes, with only minor variations. While there is some imbalance between the minimum (24) and maximum (68) samples per class, the mean around 41 samples indicates the data is reasonably balanced overall.

d. There are benchmark results available for various methods tested on this dataset, highlighting the challenges and successes of different architectures:

1. <u>State-of-the-Art Results:</u>

   Advanced models like EfficientNet, particularly the EfficientNet-B7 variant, have achieved impressive accuracy rates on the Stanford Cars dataset. For example, the Mish EfficientNet-B7 combined with the Ranger optimizer reached 94.79% accuracy in tests across multiple runs. This result is very close to the current state-of-the-art performance, which is approximately 96% using domain-adaptive transfer learning.

   These results demonstrate the effectiveness of leveraging modern deep learning techniques and powerful pre-trained architectures for complex datasets.

2. <u>Performance of Popular Architectures:</u>

   ResNet architectures such as ResNet50 and ResNet152, as well as DenseNet201, have been widely used for fine-tuning on the Stanford Cars dataset. These models typically achieve accuracies in the range of 85% to 90%, reflecting their ability to learn intricate patterns when trained on this dataset.

   However, achieving such results often requires significant fine-tuning, careful selection of hyperparameters, and additional data augmentation.

3. <u>Limitations of Regular CNNs:</u>

   Simpler CNN architectures, such as those with fewer layers or filters, often fail to generalize well on this dataset. Although these models may show low training loss and reasonable performance on validation data, their test accuracy is typically poor.

   This discrepancy suggests that regular CNNs lack the capacity to capture the fine-grained and complex patterns required to differentiate between the highly similar car classes in the Stanford Cars dataset.

   Observations from various implementations confirm that small or non-complex networks perform inadequately, struggling with overfitting despite achieving low training loss.

2. Model Forming neural network:

a. The implemented model begins with a preprocessing pipeline that prepares the input images for the CNN architecture. Images are resized to 224×224 ensuring uniformity and compatibility with the network. Each image is normalized to a range of
[−1,1] using mean and standard deviation values of 0.5 for all three RGB channels. This preprocessing stabilizes the training process and ensures consistent input scaling.

The CNN architecture itself has a well-defined structure with two main parts: convolutional layers and fully connected layers. The convolutional layers are organized into five blocks, each consisting of the following sequence:

1. A Conv2d layer that applies filters to extract feature maps:
   - Block 1: 16 filters.
   - Block 2: 32 filters.
   - Block 3: 64 filters.
   - Block 4: 128 filters.
   - Block 5: 256 filters.

2. A BatchNorm2d layer after each convolution to normalize activations and improve training stability.
3. A ReLU activation function to introduce non-linearity and allow the network to model complex patterns.
4. A MaxPool2d layer with a kernel size of 2 and stride of 2, which halves the spatial dimensions to retain only the most significant features.

After the convolutional layers, the network transitions to fully connected layers:

1. The flattened output of the convolutional layers (size 12544) is passed through fully connected layers with progressively smaller dimensions:
    - 12544→8192→1024→512→256→196
2. Each fully connected layer is followed by:
    - A BatchNorm1d layer for activation normalization.
    - A ReLU activation function.
    - A Dropout layer with rates ranging from 0.3 to 0.5 for regularization.

This systematic architecture efficiently captures hierarchical features while addressing overfitting through normalization and dropout.

b. The training process was carried out using 5-fold cross-validation, where each fold contributed towards evaluating the robustness of the model in different data splits. The analysis focuses on comparing the model's performance across training, validation, and test sets for each fold.

Loss and Accuracy Plots

The plots present the metrics for each of the five folds:

Loss Plot (Training, Validation, Test)

- The training loss consistently decreased throughout the epochs, indicating a gradual improvement of the model on the training data.
- The validation loss decreased at a slower rate compared to the training loss, and showed some fluctuation, particularly in the first few epochs. This suggests that the model is learning, but some overfitting might be occurring as the difference between training and validation losses is significant towards the end.

Accuracy Plot (Training, Validation, Test)

- The training accuracy consistently improved and reached high values (close to 1.0) for all folds, demonstrating that the model was able to fit the training data well.
- Validation accuracy, however, remained relatively low throughout the training process, and did not see significant improvements beyond a certain point, suggesting underfitting on validation data or that the model struggles to generalize.

Per-Fold Metrics Analysis

For each fold, training accuracy reached around 98-100% by the end of training, while validation accuracy fluctuated between 20-25%, and the test accuracy hovered around similar values.

- Training vs. Validation/Test Comparison:

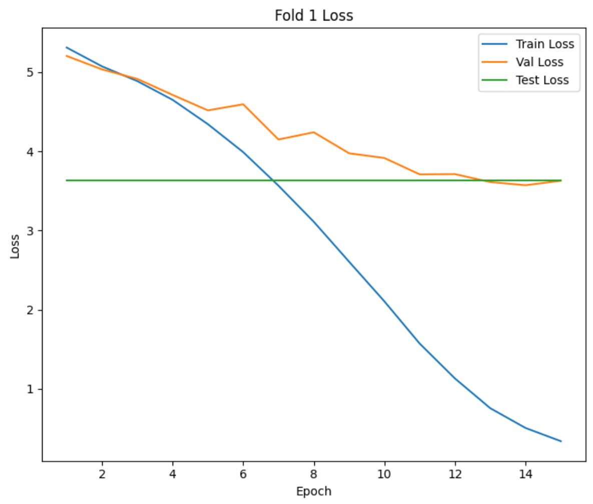
  A significant gap is evident between the training and validation/test metrics, especially in accuracy. This gap suggests overfitting—where the model learns the training data too well, including noise and irrelevant details, which affects its ability to generalize to new data.

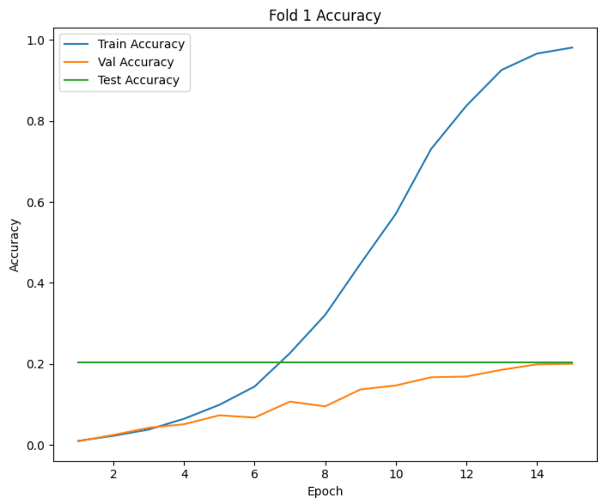
  The model's loss metrics also reflect this, with the training loss reaching much lower values compared to validation and test loss.

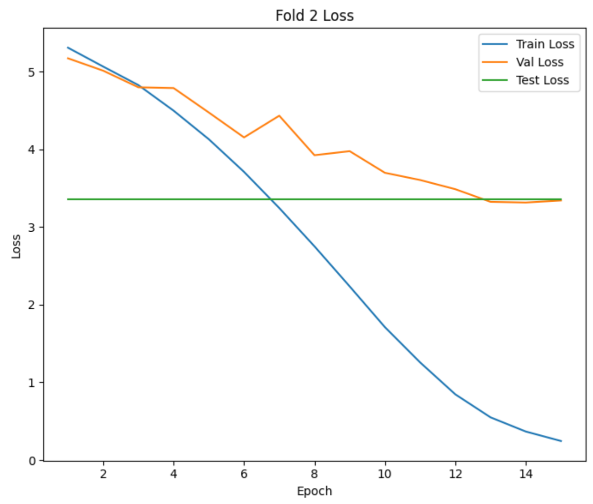
- Mean Accuracy Across Folds:

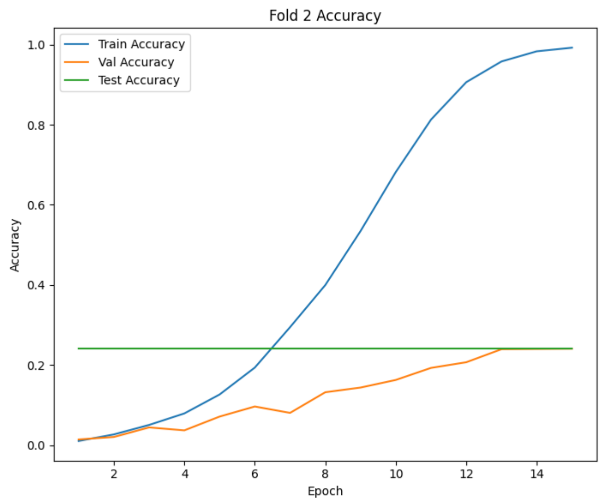
  The mean accuracy plot across all folds further highlights the difference between training and validation/test performances.

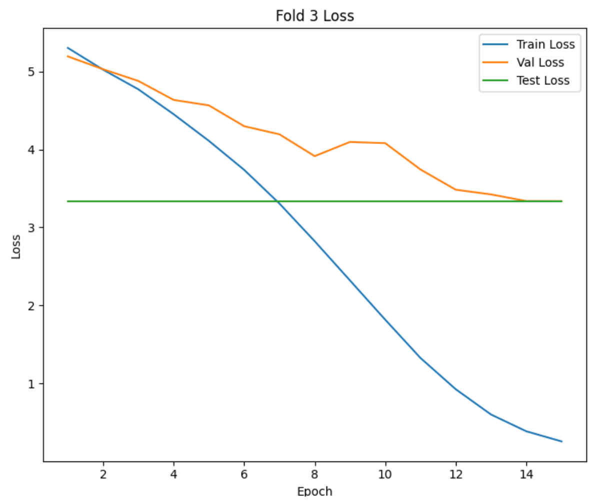
  While training accuracy reaches almost perfect values, validation and test accuracies remain significantly lower, both stabilizing at around 20-25%. This means that even after averaging across different splits of the data, the model consistently struggles to achieve good performance on unseen data.

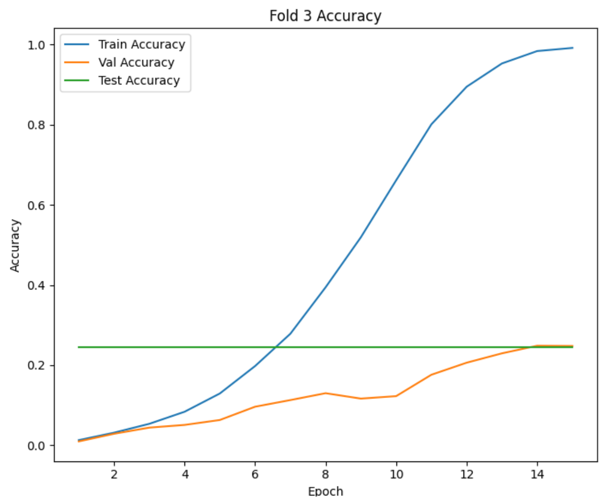

Fold 1 Loss
Train Loss
Val Loss
Test Loss
Loss
Epoch

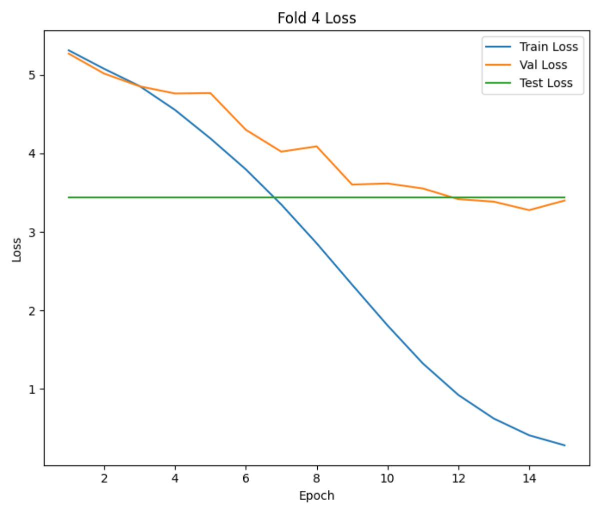

Fold 1 Accuracy
Train Accuracy
Val Accuracy
Test Accuracy
Accuracy
Epoch

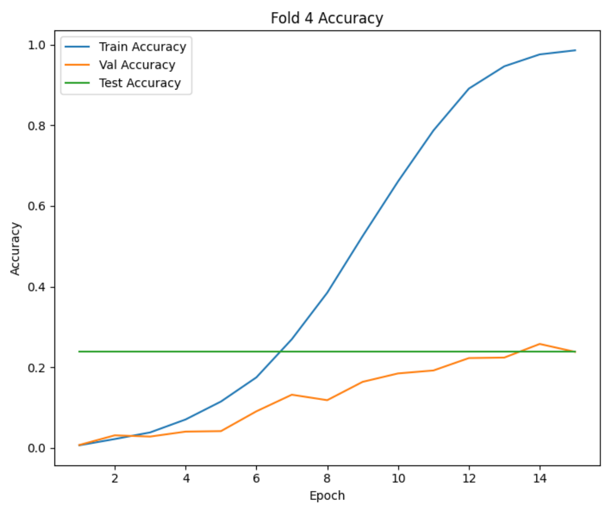

Fold 2 Loss
Train Loss
Val Loss
Test Loss
Loss
Epoch

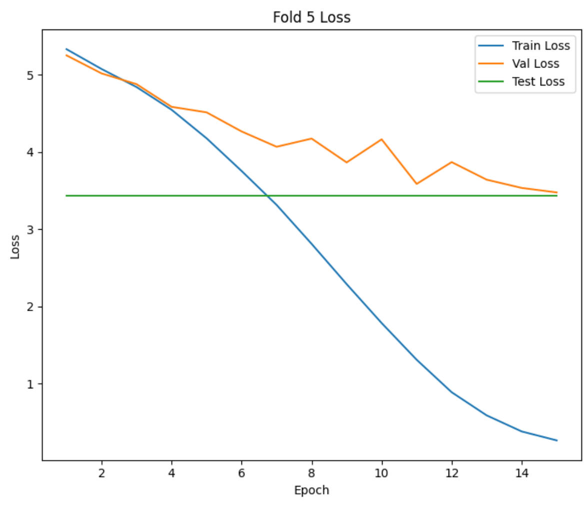

Fold 2 Accuracy
Train Accuracy
Val Accuracy
Test Accuracy
Accuracy
Epoch

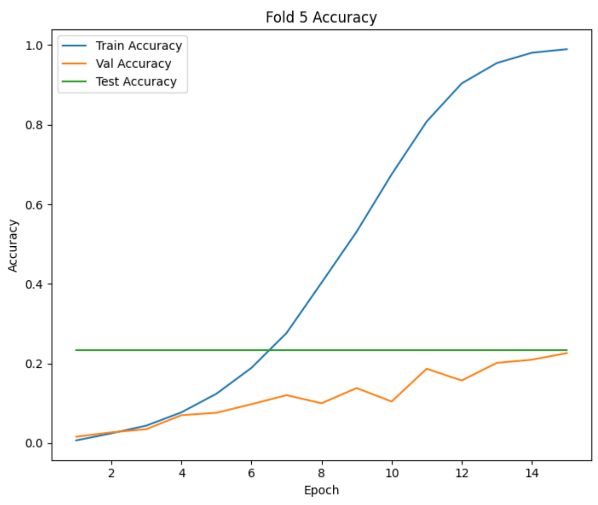

Fold 3 Loss
Train Loss
Val Loss
Test Loss
Loss
Epoch

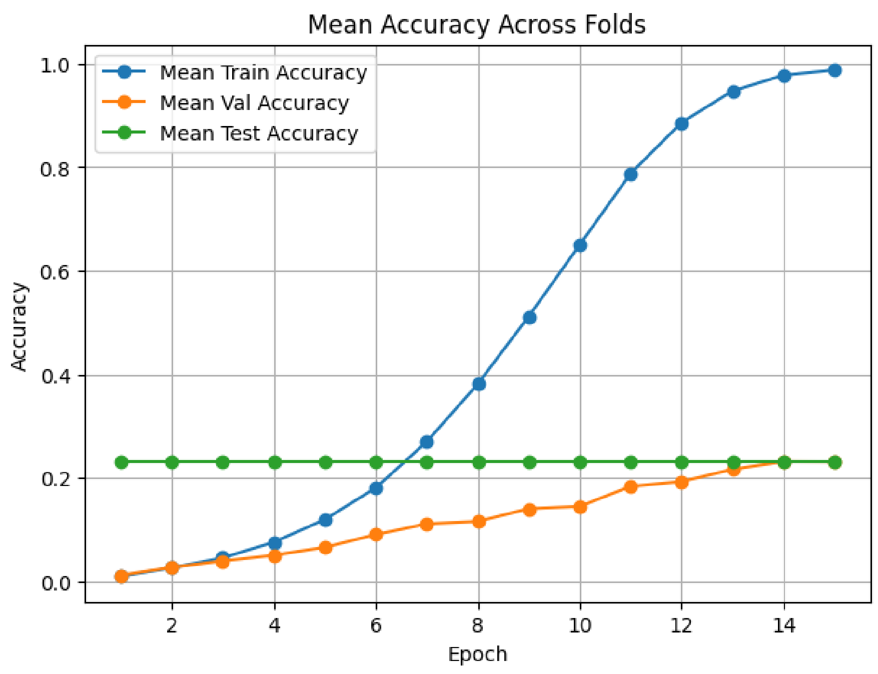

Fold 3 Accuracy
Train Accuracy
Val Accuracy
Test Accuracy
Accuracy
Epoch



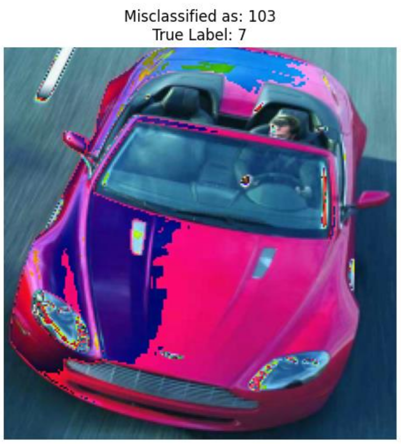


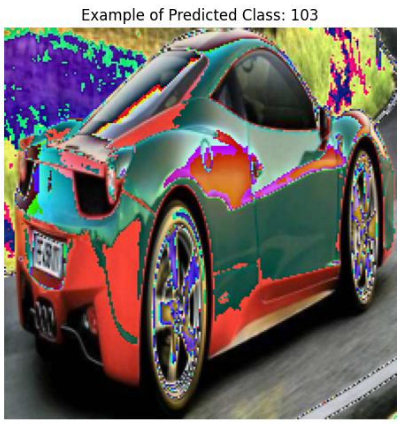


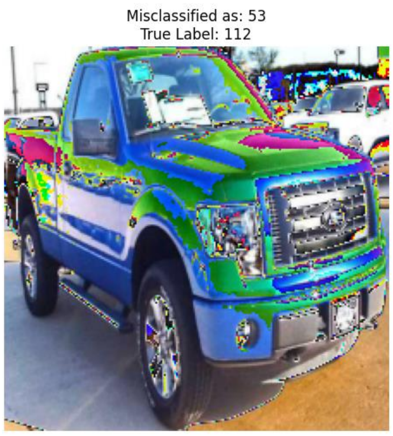


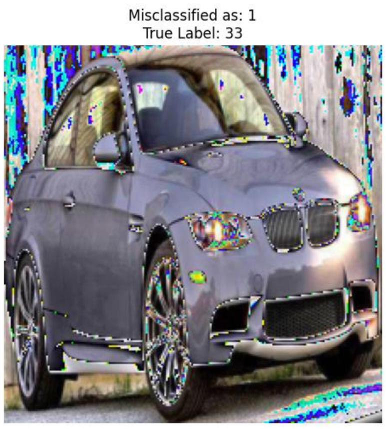


Bad cllasifications:









Possible Reasons for Misclassifications

1. **Ambiguous Features**

    - The baseline model may struggle with distinguishing between classes that share visually similar features. For example, different car models with similar shapes or colors might confuse the model, leading to incorrect classifications.

    - A detailed analysis of misclassified examples could reveal patterns of similarity that the model fails to capture effectively.

2. **Dataset Quality**

    - Issues such as low-quality images, mislabeled samples, or noise in the dataset can negatively impact the model's ability to learn robust features.

   - Cleaning the dataset and verifying labels could improve performance by ensuring that the model learns from accurate and high-quality data.

3. **Overfitting**

   - The baseline model shows a significant gap between training and validation/test performance, suggesting overfitting. This occurs when the model memorizes specific patterns in the training data rather than learning generalizable features.

   - Overfitting might stem from insufficient regularization, lack of data augmentation, or overly complex fully connected layers.

Proposed Improvements and Their Implementation in the Improved Model

1. **Regularization Techniques**

   - Introduced **L2 regularization** during training to penalize large weights and encourage simpler solutions.
   - Maintained dropout rates but combined them with L2 regularization for better control over overfitting.

2. **Data Augmentation**

   - Implemented extensive data augmentation techniques, including random horizontal flips, rotations, and color jitter.
   - This increases dataset variability, helping the model generalize better to unseen data.

3. **Residual Blocks**

   - **Baseline**: The convolutional layers were sequential, without any skip connections or advanced mechanisms to handle deeper optimization challenges.
   - Added **residual blocks** to enable learning residual mappings (F(x) + x). This simplifies optimization and mitigates the vanishing gradient problem.
     Residual blocks allow the improved model to capture more complex features while maintaining gradient stability.

4. **Learning Rate Monitoring**

   - Added dynamic learning rate scheduling (ReduceLROnPlateau) based on validation loss to improve convergence.

c. **New Architecture: CARS_IMPROVED_CNN:**

The **CARS_IMPROVED_CNN** architecture builds upon the initial model by introducing several enhancements in the convolutional layers, regularization techniques, and overall structure.

Convolutional Layers with Residual Blocks

1. **Sequential Convolutional Layers**:

   - The convolutional layers progressively extract spatial and feature representations from the input images. Each convolution is followed by:

     - Batch normalization to stabilize learning.

     - ReLU activation for non-linearity.

     - Max pooling for down-sampling.

2. **Residual Blocks**:
    - Two **residual blocks** are integrated into the architecture:
        - **First Residual Block**: Applied to 32-channel feature maps.
        - **Second Residual Block**: Applied to 64-channel feature maps.
    - **Mechanism**: Each block contains two convolutional layers, batch normalization, and ReLU activations. A **skip connection** adds the input of the block to its output (F(x) + x), and the combined result passes through another ReLU activation.

Fully Connected Layers

After feature extraction, the output from the convolutional layers is flattened and passed through fully connected layers:

1. **Layer Structure**:
    - A sequence of linear layers interspersed with:
        - Batch normalization to maintain stability and avoid vanishing gradients.
        - ReLU activations for non-linear transformations.
        - Dropout layers to prevent overfitting by randomly deactivating neurons.
    - The final layer outputs logits corresponding to the 196 classes.

Regularization and Optimization

1. **Regularization**:
    - **Dropout**: Introduced at different levels of the fully connected layers.
    - **L2 Regularization**: Applied during training to penalize large weights, ensuring smoother and simpler models.
2. **Dynamic Learning Rate Scheduling**:
    - **ReduceLROnPlateau**: Adjusts the learning rate based on validation loss, ensuring steady progress during training.

Data Augmentation

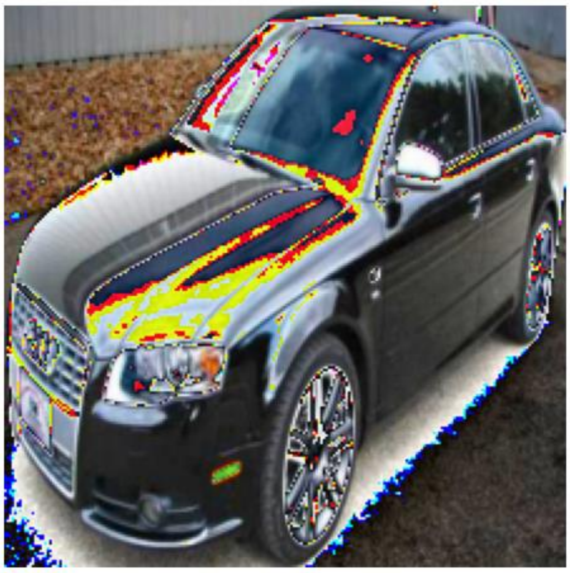
- **Training Data**: Augmented with horizontal flips, rotations, and color jitter. This simulates variability in the dataset, reducing overfitting and enhancing generalization.
- **Testing Data**: Standard resizing and normalization to maintain consistency.

Miss classification:

Misclassified as: 96
True Label: 91

Example of Predicted Class: 96

Misclassified as: 48
True Label: 23



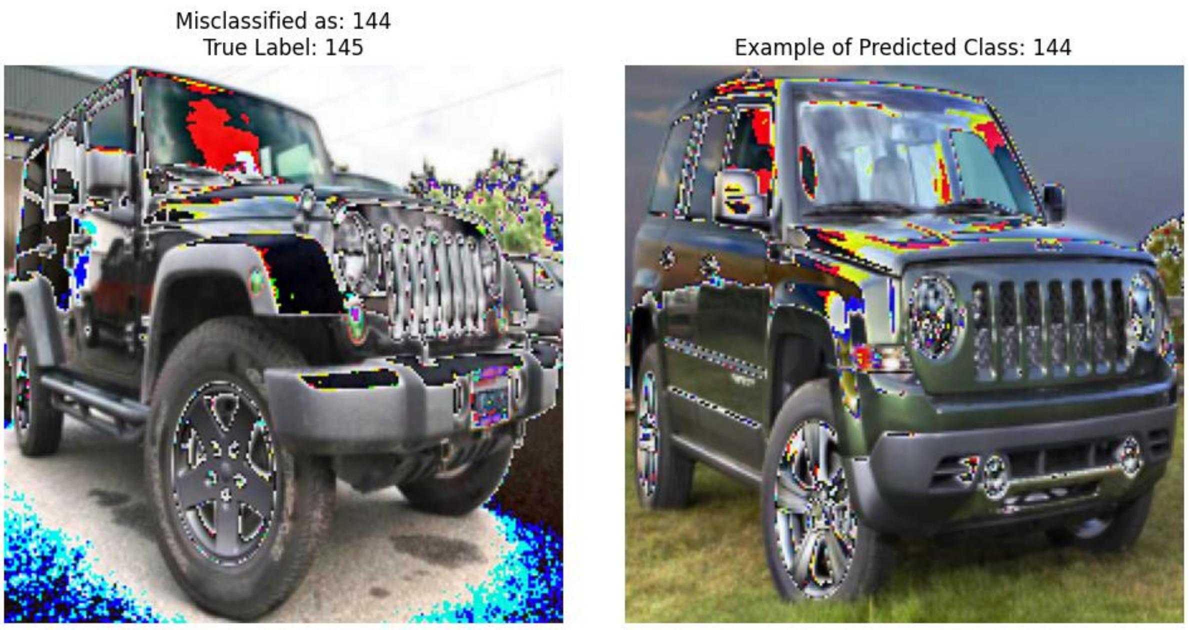
Example of Predicted Class: 48

Misclassified as: 123
True Label: 124

Example of Predicted Class: 123

Misclassified as: 180
True Label: 182

Example of Predicted Class: 180



Observed Misclassifications

Despite significant improvements in the architecture and training process, the model still exhibits misclassifications. Possible reasons:

1. **Class Overlap or Ambiguous Features**
    - Some car models may share highly similar visual features, such as shape, color, or other design aspects, making them difficult to distinguish.
    - This overlap in features leads to confusion, especially in scenarios where the dataset does not have sufficient distinguishing characteristics for these classes.
2. **Dataset Limitations**
    - **Image Quality**: Low-resolution, blurry, or occluded images may prevent the model from learning meaningful features effectively.
3. **Residual Block Effectiveness**
    - Although residual blocks help in learning complex features, they could lead to insufficient regularization if the dataset lacks diversity. This might cause the model to focus too much on subtle patterns in the training data that do not generalize well.

Suggested Improvements for model

1. **Implement Class-Specific Regularization**
    - Fine-tune the model on classes with higher misclassification rates to help balance the performance across categories.
2. **Robust Augmentation During Training and Inference**
    - Add more extensive augmentation techniques during training, such as random perspective shifts or blurring, to mimic real-world variations.
3. **Feature-Specific Attention Mechanisms**
    - Introduce **attention mechanisms** (e.g., self-attention) to allow the model to focus on the most relevant parts of the image, which may help in distinguishing ambiguous classes.
4. **Optimize Residual Block Placement**

- Experiment with the placement and configuration of residual blocks to ensure they are optimally positioned for feature extraction in deeper layers. For example, introducing additional residual connections in later stages might better capture global features.

3.

Model Name	# parameters	Validation Loss	Validation Accuracy	Test Loss	Test Accuracy	# unique correct samples	# unique errors
1. resnet50	~23.5M	0.3889	0.8926	0.3710	0.9029	7260	781
2. vgg16	~134.3M	0.7390	0.8140	0.7313	0.8184	6581	1460
3. densenet121	~7.0M	0.6807	0.8711	0.6425	0.8866	7129	912
4. efficientnet_b0	~4.0M	0.9183	0.7802	0.8850	0.7933	6379	1662

d. **Results Comparison: ResNet50 vs. Random Forest Classifier**
ResNet50 Performance:

- **Accuracy**: 90.29%
- **Test Loss**: 0.3710

Random Forest Classifier (Feature Extractor from ResNet50):

- **Accuracy**: 89.00%
- **Macro Average (Precision, Recall, F1)**: 88-89%
- **Weighted Average (Precision, Recall, F1)**: 89%

Performance Comparison:

- The ResNet50 model slightly outperforms the Random Forest classifier in accuracy (90.29% vs. 89.00%).
- The deep learning model shows better error handling, with fewer unique errors (781 vs. ~880 from Random Forest assuming the accuracy difference translates proportionally).

Using ResNet50 as a feature extractor for Random Forest shows the power of combining deep learning for feature extraction with classical machine learning for classification.
While the Random Forest classifier offers simplicity and competitive results, the fully trained ResNet50 model remains superior in terms of accuracy and robustness.

Experiment	Description	Runtime	Loss	Additional Metrics	Parameter Settings	Main processing changes	Other significant changes
Baseline Model	Built an initial CNN with 5 convolutional layers, batch normalization, max-pooling, and fully connected layers for classification.	~1.5 hours	Cross-Entropy Loss train- 0.338 Val- 3.630	Accuracy and Validation Loss	Batch size: 128, Learning rate: 0.0005, Max epochs: 15	- Initial setup with standard normalization and resizing of the dataset to 224x224.	
K-Fold Cross Validation base model	Assessed model performance using 5-fold cross-validation with standardized data splitting and varying hyperparameters to evaluate consistency and robustness across folds.	~3 hours	Test Loss (Avg): 3.44 Validation Loss (Avg): 3.44	Test accuracy: ~23-24%, Train accuracy: ~99%	Batch size: 128, Learning rate: 0.0005, Max epochs: 15	Implemented K-Fold cross-validation; separate dataloaders for training, validation, and testing; metrics logged across epochs and folds.	Results visualized with fold-specific and mean accuracy/loss plots.
Analysis of Misclassifications	Analyzed misclassified samples to understand patterns of errors, focusing on ambiguous features, class imbalance, and dataset noise. Insights guided improvement strategies.	~1 hour	-	Misclassified examples identified; visualized incorrect predictions.	-	Filtered misclassified test samples to visualize and analyze errors; identified common failure patterns such as visually similar classes and dataset issues.	Suggested changes include data augmentation, regularization, and adjusting class weights to address misclassification patterns.
Improved Network Implementation	The improved network added regularization techniques, including dropout and L2 regularization, as well as batch normalization to enhance generalization. Data augmentation was employed for training, and the model architecture was improved with additional layers and residual blocks.	~1 hour	Final Test Loss: 2.089	Test Accuracy: 45.43%	Batch Size: 128 Max Epochs: 25 Learning Rate: 5e-4 Optimizer: Adam with weight decay (1e-4)	Data augmentation added: horizontal flips, rotations, color jittering Dropout layers (0.3-0.5) added Batch normalization applied Residual blocks implemented L2 regularization used	Model size increased to 112M trainable parameters. Validation accuracy plateaued at 41.7%, suggesting further improvements could be made with advanced techniques.

Cross-Validation on Improved Net	Training completed for 5 folds with each fold taking ~25 seconds per epoch, 25 epochs in total per fold.	~ 4 hours	Average Test Loss: 2.21	Average Test Accuracy: 43.17%	Batch Size: 128 Max Epochs: 25 K: 5 folds Learning Rate: 5e-4 Optimizer: Adam with weight decay (1e-4)	5-fold cross-validation implemented to assess the robustness and consistency of the improved network. Validation metrics logged per fold for analysis.	Minor variation observed across folds in validation/test loss and accuracy. Results indicate consistent performance with room for further optimization.
Misclassification Analysis	Analysis performed post-training using test set predictions. Runtime varies depending on dataset size (few seconds for ~8,000 images).	~ 1hour	Misclassified samples identified for further insights. Identified patterns where the network struggles to distinguish visually similar classes.	-	-	Post-test analysis focused on incorrect predictions to investigate patterns in misclassification. Visual and statistical analysis performed.	Found specific classes with high misclassification rates (e.g., similar visual patterns across classes). Insights to improve class separation include increasing dataset diversity and adjusting augmentation techniques.
Inference-Time Augmentation	Runtime increased significantly due to repeated predictions on augmented test examples (~25 predictions per test image).	Runtime increased significantly due to repeated predictions on augmented test examples (~25 predictions per test image). ~1 hour	-	Test Accuracy: 19.80%	Batch Size: 128 Max Epochs: 25 Learning Rate: 5e-4 Cross-Entropy Loss Function Num Augmentations: 25	Applied multiple augmentations at inference for each test image to enhance prediction robustness. Most common predicted class used for final output.	Test accuracy significantly decreased. Likely causes include overfitting during training and suboptimal augmentations that may distort the input images rather than generalize features. Further tuning of augmentation strategies is necessary.
Adding a New Category	102 images of the "BIG_TRUCKS" class. Updated the dataset, loaders, and the final fully connected layer to accommodate 197 classes.	~1 hour	Test Loss: 2.1154	Test Accuracy: 44.97% BIG_TRUCKS Accuracy: 98.04% Misclassified BIG_TRUCKS: 2	Batch Size: 128 Max Epochs: 25 Learning Rate: 5e-4 Cross-Entropy Loss Function 197 Classes in Output Layer	High accuracy achieved for the new class. However, overall test accuracy remains low. Indicates effective learning for the new category but potential overfitting or under-generalization for other classes. Further balancing of dataset needed. -	

Transfer Learning	Fine-tuned four popular pretrained models to classify 196 car classes. Modified the output layer to match the dataset's 196 classes. Each model was trained and validated independently.	~30 minutes per model training; <10 minutes per test	ResNet50: 0.3710 VGG16: 0.7313 DenseNet121: 0.6425 EfficientNet-B0: 0.8850	ResNet50: 90.29% VGG16: 81.84% DenseNet121: 88.66% EfficientNet-B0: 79.33%	Pretrained models: ResNet50, VGG16, DenseNet121, EfficientNet-B0 Batch Size: 128 Epochs: 10 Learning Rate: 1e-4	ResNet50 achieved the highest test accuracy (90.29%) and lowest test loss (0.3710). The VGG16 model underperformed compared to others, with higher test loss and lower accuracy. Pretrained models demonstrated strong transfer learning benefits. -
Feature Extraction and Classifier	Utilized the trained ResNet50 model to extract features by removing the last fully connected layer. Features were used to train a Random Forest Classifier for improved generalization.	~15 minutes	--	Random Forest Test: 88.72%	ResNet50 model (trained); Random Forest: n_estimators=100, random_state=42	ResNet50 provided robust features, enabling the Random Forest to achieve competitive accuracy. Feature extraction streamlined the model pipeline. -

4. In this assignment, we explored the potential of deep learning for classifying car models using the Stanford Cars dataset. Our approach began with designing and implementing a baseline convolutional neural network (CNN) consisting of five convolutional blocks and fully connected layers. This baseline served as a foundation to experiment with techniques such as dropout and batch normalization to mitigate overfitting and stabilize training. We evaluated the model using k-fold cross-validation, providing insights into its robustness across different splits of the data.

Additionally, we leveraged transfer learning by fine-tuning pre-trained architectures, including ResNet50, DenseNet121, EfficientNet-B0, and VGG16, adapting them to our dataset. The results revealed nuanced differences in the models' capabilities, with ResNet50 demonstrating the highest accuracy of 90.29%, highlighting its effectiveness in extracting intricate features. Furthermore, we explored the integration of deep learning with classical machine learning by using ResNet50 as a feature extractor for a Random Forest classifier, achieving a competitive accuracy of 89.00%. This hybrid approach provided valuable insights into the trade-offs between modern deep learning architectures and traditional methods in tackling fine-grained classification challenges.