Group 3 - Jiota, Gregor, Michalis

Comparative Analysis of Custom Convolutional Neural Network and Pre-trained ResNet50 on CIFAR-10 Dataset

For this project, we were given the choice between the CIFAR-10 and Animals-10 datasets. After careful consideration, we selected the CIFAR-10 dataset due to its manageable complexity and suitability for our objectives. CIFAR-10 is a well- established benchmark dataset that provides a balanced mix of complexity and accessibility, making it particularly advantageous for experimenting with and comparing different convolutional neural network (CNN) architectures. Additionally, its relatively smaller image size and straightforward nature allowed us to focus on the architectural nuances and performance tuning of our models, ultimately facilitating a more efficient and insightful comparison.

Introduction

The goal of this project is to develop a convolutional neural network (CNN) model to classify images from the CIFAR-10 dataset into ten predefined categories. CNNs are particularly effective for image classification problems due to their ability to automatically learn spatial hierarchies of features through convolutional layers, which makes them well-suited for tasks involving visual data. We utilised two different CNN models for this purpose: a custom CNN model built entirely from scratch and a pre-trained ResNet50 model, leveraging the power of transfer learning. This report provides a comprehensive overview of the approach taken, detailing the preprocessing steps, model architectures, training process, and performance evaluation. Additionally, we highlight the insights gained during experimentation and compare the two models to determine the most effective architecture for this classification task.

1. Data Preprocessing

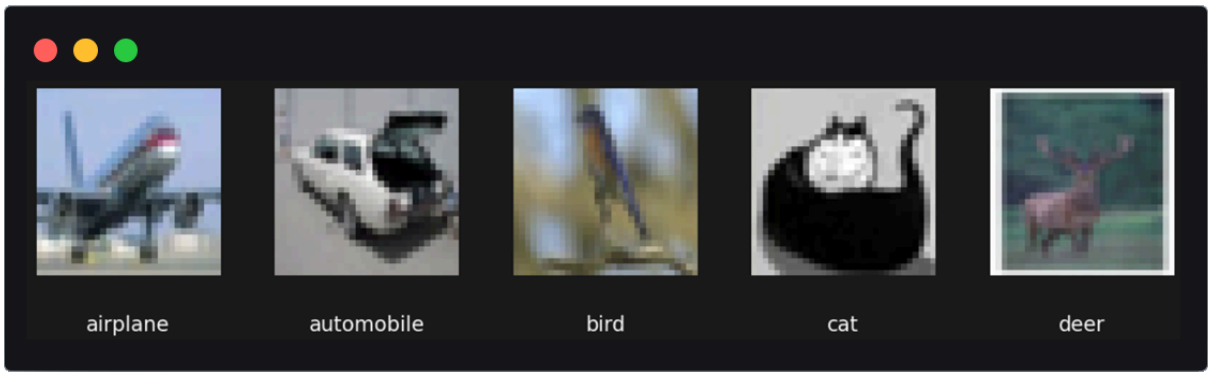


Figure 1 | `Display of sample images from the CIFAR-10 training dataset`

Dataset Overview

The CIFAR-10 dataset consists of 60,000 images, each of size 32x32, belonging to ten different classes: aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. One of the main challenges associated with this dataset is its relatively small image size, which can make it difficult for models to extract detailed features.

Additionally, the diversity of the classes adds complexity, as the model must learn to differentiate between a wide range of objects, some of which share visual similarities (e.g., cats and dogs). The dataset is split into a training set of 45,000 images, a test set of 10,000 images (which represents 10% of the entire dataset), and a validation set of 5,000 images.

The CIFAR-10 dataset was loaded and converted into NumPy arrays. The images were resized from their original dimensions of 32x32 to 64x64 for the custom CNN and to 96x96 for the ResNet50 model respectively.

One-hot Encoding: The labels were one-hot encoded, resulting in vectors representing each of the 10 classes.

Normalisation: The images were normalised by dividing each pixel value by 255, thereby rescaling them between 0 and 1. This helps improve model convergence during training.

Data Augmentation: Data augmentation was applied using the ImageDataGenerator class, including horizontal and vertical flipping, zoom, width and height shifts, and rotation (Figure 2). This increased the diversity of the training data and helped prevent overfitting.

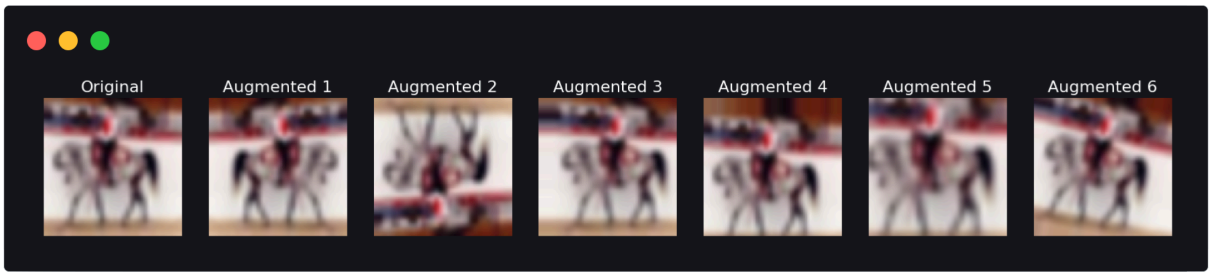


Figure 2 | Data augmentation example on the sample class "horse" with `horizontal_flip, vertical_flip, width_shift_range=0.2, height_shift_range=0.2, zoom_range=0.2,`and `rotation_range=15.`

2. Model Architectures

Custom CNN

The custom CNN model was designed from scratch to effectively capture features from the CIFAR-10 dataset while maintaining a balance between model complexity and training efficiency. The architecture was composed of five main convolutional blocks, followed by fully connected layers (Figure 3). The detailed architecture is as follows:

- **Input Layer:** The input layer was defined with a shape of (64, 64, 3) to accommodate the resized CIFAR-10 images.
- **Convolutional Blocks:** The model consisted of five convolutional blocks, each designed to progressively learn more abstract and high-level features from the input images:
    1. **First Block:** The first block consisted of two Conv2D layers, each with 128 filters of size (3, 3), followed by a BatchNormalization layer to normalise the activations and improve stability during training. A MaxPooling2D layer was

used to reduce the spatial dimensions, and a Dropout layer with a rate of 0.2 was added to prevent overfitting.

2. **Second Block:** Similar to the first block, the second block included two Conv2D layers with 128 filters, BatchNormalization, MaxPooling2D, and a Dropout layer with an increased rate of 0.3. This block further extracted intermediate-level features while adding regularisation to prevent overfitting.
3. **Third Block:** The third block increased the number of filters to 256, allowing the model to capture more complex patterns. It consisted of two Conv2D layers, followed by BatchNormalization, MaxPooling2D, and a Dropout layer with a rate of 0.4.
4. **Fourth Block:** The fourth block further increased the filter count to 512, enabling the model to learn highly abstract features. It included two Conv2D layers, BatchNormalization, MaxPooling2D, and a Dropout layer with a rate of 0.5.
5. **Fifth Block:** The fifth block also had 512 filters, similar to the fourth block. It consisted of two Conv2D layers, BatchNormalization, MaxPooling2D, and a Dropout layer with a rate of 0.5. This block aimed to refine the high-level feature representations before passing them to the fully connected layers.

All the blocks contain “same” padding which ensures that the spatial dimensions of the output feature maps are the same as the input, and help preserve the information at the borders of the image. A ReLU activation function was applied after each convolution to introduce non-linearity, allowing the model to learn more complex patterns. This activation function helps the model to capture complex relationships within the data by allowing only positive values to pass through while setting negative values to zero.

- **Fully Connected Layers:** After the convolutional blocks, the output was flattened to create a one-dimensional vector. This vector was then passed through a Dense layer with 128 neurons and ReLU activation. BatchNormalization and Dropout (rate of 0.5) were applied to improve generalisation. Finally, a Dense output layer with 10 neurons and softmax activation was used to predict the probabilities for each of the 10 classes.

The custom CNN model architecture was designed to be deep enough to capture the complexity of the CIFAR-10 dataset while incorporating several regularisation techniques (BatchNormalization and Dropout) to prevent overfitting.

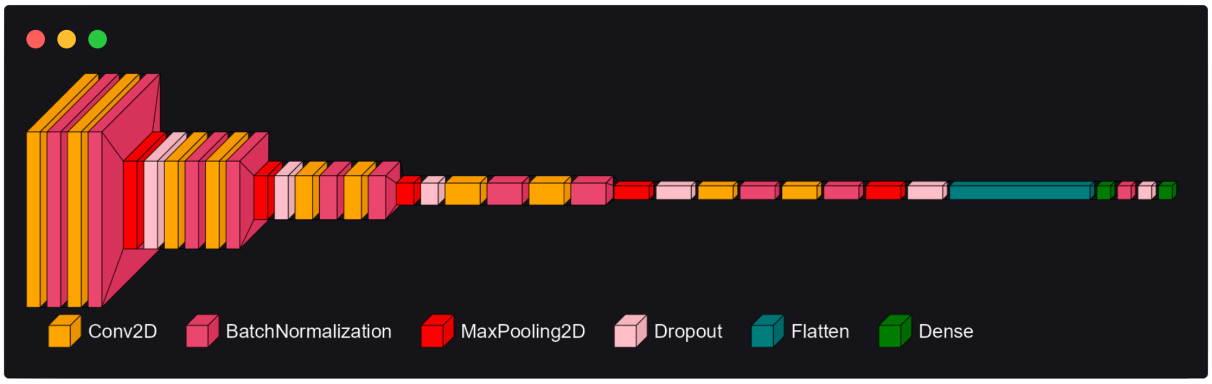


Figure 3 | Architecture of the custom CNN model

Pre-trained ResNet50

The second model employed in our experiments is based on the ResNet50 architecture, a highly sophisticated pre-trained model with weights derived from theImageNet dataset. We leveraged the ResNet50 model as a powerful feature extractor by initially freezing its convolutional base, allowing us to focus on training the added classification layers. Specifically, we appended a GlobalAveragePooling layer, followed by a fully connected Dense layer comprising 512 units and a Dropout layer to mitigate overfitting (Figure 4). The final output layer employed a softmax activation function to classify images into the ten categories. The training process was conducted over three distinct cycles, where we utilised the saved weights from the previous cycle at the beginning of each subsequent one. To enhance the model's learning capabilities progressively, we unfroze 15 additional layers in the second cycle and 30 layers in the third cycle, enabling the model to fine- tune more complex features and thereby improve its performance. The ResNet50 model consists of a total of 24,641,930 parameters, out of which 1,054,218 were trainable during the training process.

The detailed architecture is as follows:

- **Base Model:** The base model was the ResNet50 architecture, which is known for its deep residual learning capabilities. Residual networks use skip connections to allow gradients to flow more easily through the network during backpropagation, thereby addressing the vanishing gradient problem often encountered in deep networks. In this project, the ResNet50 model was loaded without the top (fully connected) layers, and the ImageNet-trained weights were retained. The input shape for the base model was set to (96, 96, 3) to accommodate the resized CIFAR-10 images.
- **Freezing Layers:** Initially, all layers of the ResNet50 base model were frozen to retain the pre-trained features. This allowed the newly added layers to train without altering the learned representations from ImageNet.
- **Custom Layers:** To adapt the pre-trained ResNet50 model for CIFAR-10 classification, custom layers were added on top of the base model:
    1. **GlobalAveragePooling2D Layer:** A GlobalAveragePooling2D layer was added to reduce the spatial dimensions of the feature maps while retaining the most important features. This pooling layer effectively reduced the output

of the convolutional base to a smaller vector, which was then passed to the fully connected layers.

2. **Dense Layers:** A Dense layer with 512 neurons and ReLU activation was added, followed by a Dropout layer with a rate of 0.3 to prevent overfitting. Finally, a Dense output layer with 10 neurons and softmax activation was used to generate class probabilities for CIFAR-10.

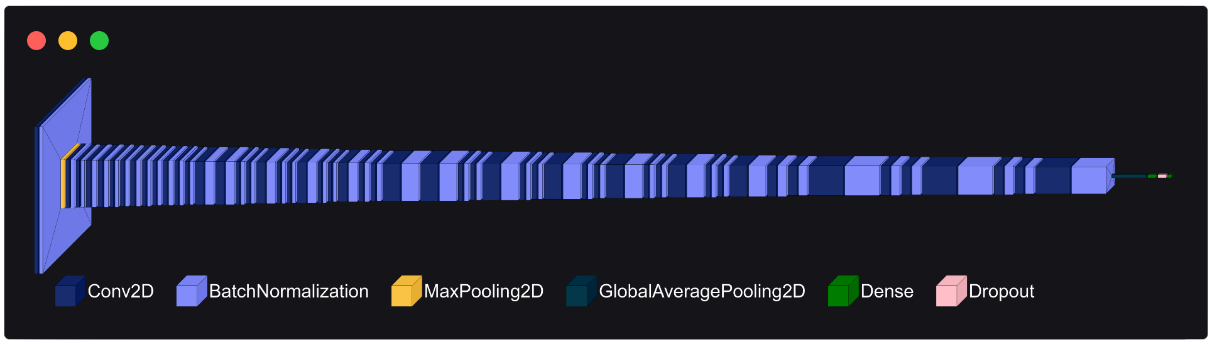


Figure 4 | Architecture of the pre-trained ResNet-50 model

The ResNet50 model's architecture leveraged the power of pre-trained weights to extract robust features from the CIFAR-10 dataset, which helped improve model performance significantly compared to training a CNN from scratch. The addition of custom layers allowed for fine-tuning specific to CIFAR-10 while retaining the strong feature extraction capabilities of the ResNet50 base model.

3. Training Details

The training process for both models involved careful selection of hyperparameters, model compilation, callbacks, and saving models for future use.

Custom CNN

- **Optimizer and Learning Rate Schedule:** The custom CNN was compiled using the Adam optimizer, which is well-suited for handling sparse gradients and offers adaptive learning rates. An initial learning rate of 0.001 was set, and an exponential learning rate decay was employed. The exponential decay reduced the learning rate at each step by a fixed factor, allowing the model to take larger steps during the early stages of training and progressively smaller steps as it converged. This technique helps the model to fine-tune the weights more effectively in later epochs, improving convergence.
- **Loss Function:** The categorical cross entropy loss function was used, which is appropriate for multi-class classification tasks like CIFAR-10. This loss function measures the difference between the true class labels and the predicted probabilities, and it helps in optimising the model to correctly classify the input images.
- **Batch Size and Epochs:** The model was trained with a batch size of 64, which balanced memory efficiency with effective gradient updates. The training process allowed for up to 100 epochs, but early stopping was applied to prevent overfitting.

The early stopping callback monitored the validation accuracy and halted training if the model did not improve for 10 consecutive epochs.

- **Data Augmentation:** To improve generalizability and reduce overfitting, data augmentation was applied during training. Augmentations such as horizontal and vertical flipping, zooming, and shifting helped in creating variations of the training images, making the model more robust to real-world scenarios.
- **Model Saving:** The best model weights were saved using the ModelCheckpoint callback, which tracked the validation accuracy and saved the model whenever it achieved a new best value. This allowed us to retain the version of the model with the highest validation performance.
- **Training History:** The training history, including metrics such as accuracy and loss for both training and validation sets, was saved for post-training analysis. This allowed us to plot the training curves and observe the learning dynamics, such as overfitting or underfitting.

Pre-trained ResNet50

- **Freezing Layers and Transfer Learning:** Initially, all layers of the ResNet50 base model were frozen to retain the features learned from the ImageNet dataset. The custom layers on top were trained to adapt these features to the CIFAR-10 classification task. Transfer learning allowed the model to leverage pre-trained features, significantly speeding up the convergence process and improving accuracy compared to training from scratch.
- **Optimizer and Learning Rate:** The Adam optimizer was used, with a learning rate of 0.0001. Given the complexity of the ResNet50 model, a smaller learning rate was chosen to ensure stable updates and to avoid large changes to the pre-trained weights. Additionally, a learning rate reduction strategy was applied using the ReduceLROnPlateau callback. This callback monitored the validation loss, and if the loss plateaued for a few epochs, it reduced the learning rate by a factor of 0.5. This strategy ensured that the model could make finer adjustments when progress slowed, thereby improving overall performance.
- **Early Stopping:** Similar to the custom CNN, early stopping was applied to prevent overfitting. The patience parameter was set to 10 epochs, meaning that if the model did not show improvement on the validation set for 10 consecutive epochs, training would be halted.
- **Model Saving and Loading:** The best performing ResNet50 model was saved during training, allowing us to load the model with the highest validation accuracy for evaluation on the test dataset. This step is critical in ensuring that the version of the model used for inference is the one that generalises best.
- **Training History:** The training history for the ResNet50 model was also saved and subsequently loaded for analysis. By plotting the accuracy and loss curves, we were able to assess the impact of transfer learning and observe how quickly the model converged compared to the custom CNN.

4. Results and Analysis

A. Custom CNN

Test Metrics: On the test dataset, the model achieved:

- Custom-CNN Model accuracy: 0.9202
- Custom-CNN Model precision: 0.9209
- Custom-CNN Model recall: 0.9202
- Custom-CNN Model F1 score: 0.9200

Training Metrics: The custom CNN achieved a final training accuracy of 0.9272 and a validation accuracy of 0.9660 after completing all 100 epochs. This is not a typical behaviour and can be interpreted as follows:

1. Good Performance: Both accuracies above 90% suggest that the model has learned to classify the images quite well.

Unusual Pattern: Typically, we expect training accuracy to be higher than validation accuracy. The reverse situation here indicate a few things:

a. Regularization: strong regularisation techniques (like dropout or L2 regularisation) might be more active during training than during validation, leading to this pattern.

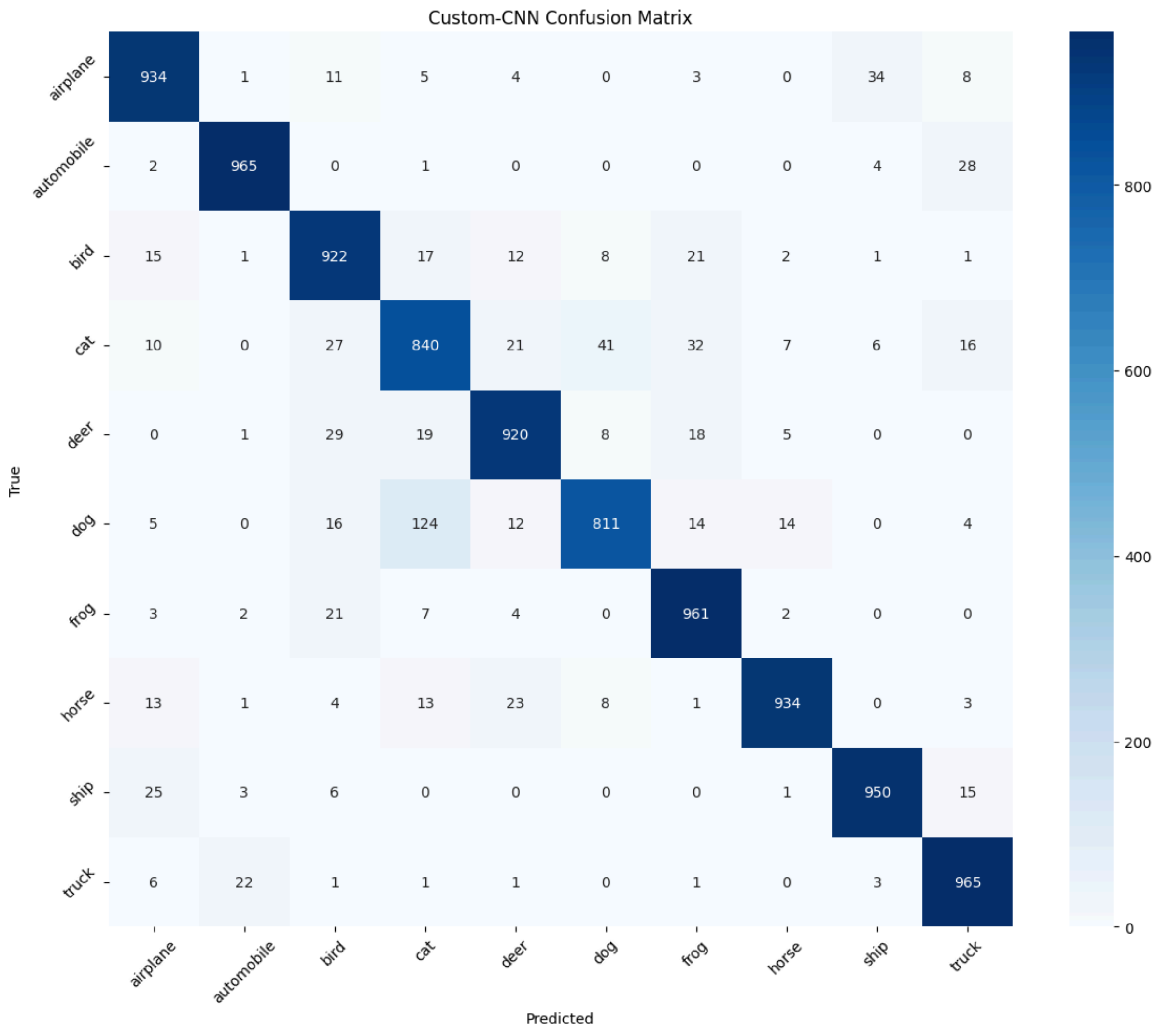
b. Data Augmentation: data augmentation was set on the training set but not on the validation set, making the training task "harder" and resulting in lower training accuracy.

c. BatchNormalization: while using batch normalisation, the stochasticity during training vs. the deterministic behaviour during validation can contribute to this pattern.

No Overfitting: The fact that validation accuracy is higher than training accuracy strongly suggests that the model is not overfitting. In fact, it might be slightly underfitting on the training data.

Generalization: The high validation accuracy suggests that the model is generalising well to unseen data.

Confusion Matrix: it revealed an overall good performance as well as that the model struggled with certain classes like 'cat' and 'dog', which are visually similar, leading to higher misclassification rates.



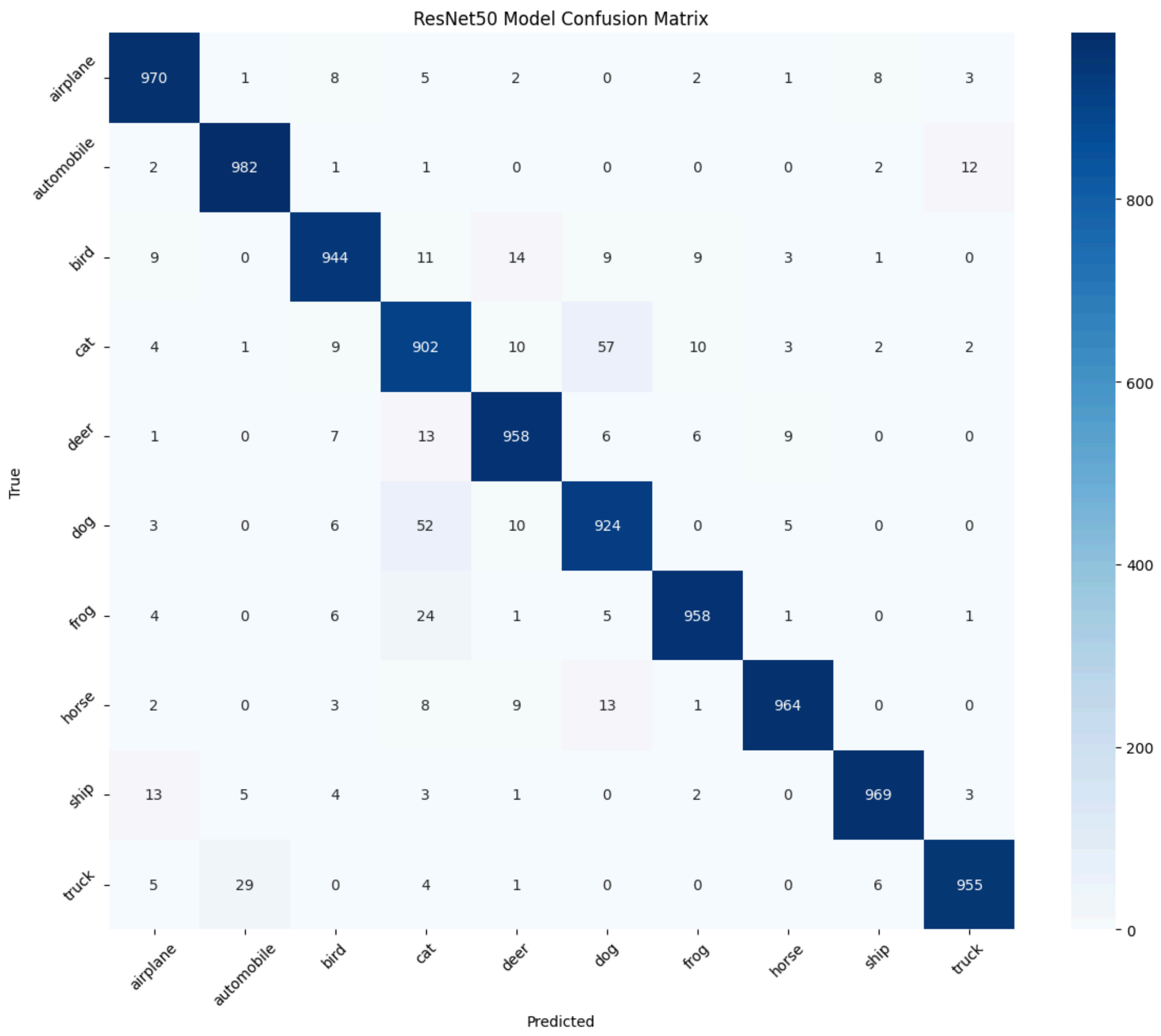
True \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	934	1	11	5	4	0	3	0	34	8
automobile	2	965	0	1	0	0	0	0	4	28
bird	15	1	922	17	12	8	21	2	1	1
cat	10	0	27	840	21	41	32	7	6	16
deer	0	1	29	19	920	8	18	5	0	0
dog	5	0	16	124	12	811	14	14	0	4
frog	3	2	21	7	4	0	961	2	0	0
horse	13	1	4	13	23	8	1	934	0	3
ship	25	3	6	0	0	0	0	1	950	15
truck	6	22	1	1	1	0	1	0	3	965

B. Pre-trained ResNet50

Test Metrics: The ResNet50 model achieved the following metrics on the test dataset:

- ResNet50 Model accuracy: 0.9526
- ResNet50 Model precision: 0.9529
- ResNet50 Model recall: 0.9526
- ResNet50 Model F1 score: 0.9527

Confusion Matrix: The confusion matrix for ResNet50 showed strong performance across all classes, while highlighting the limitations in distinguishing between similar classes, such as "cats" and "dogs".



True \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	970	1	8	5	2	0	2	1	8	3
automobile	2	982	1	1	0	0	0	0	2	12
bird	9	0	944	11	14	9	9	3	1	0
cat	4	1	9	902	10	57	10	3	2	2
deer	1	0	7	13	958	6	6	9	0	0
dog	3	0	6	52	10	924	0	5	0	0
frog	4	0	6	24	1	5	958	1	0	1
horse	2	0	3	8	9	13	1	964	0	0
ship	13	5	4	3	1	0	2	0	969	3
truck	5	29	0	4	1	0	0	0	6	955

5. Best Model

The ResNet50 model outperformed the custom CNN, achieving a higher accuracy, precision, recall, and F1 score. This superior performance can be attributed to several key factors:

1. **Deeper Architecture:** ResNet50 is significantly deeper than the custom CNN, with 50 layers compared to the relatively shallow architecture of the custom model. The deeper architecture allows ResNet50 to learn more complex and abstract features, which contributes to its improved performance. The use of skip connections in ResNet50 also mitigates the vanishing gradient problem, enabling the model to train effectively even with a large number of layers. These skip connections help the model maintain the flow of information, resulting in better gradient propagation and a more stable training process.

2. **Pre-training on ImageNet:** The ResNet50 model was pre-trained on the large-scale ImageNet dataset, which contains millions of labelled images spanning a wide variety of classes. This pre-training allowed the model to learn general features such as edges, textures, and shapes, which are transferable to many other image classification tasks, including CIFAR-10. By leveraging these pre-trained features, the ResNet50 model was able to achieve high performance with relatively less training on CIFAR-10 compared to training from scratch. Essentially, the pre-training provided a strong feature extraction foundation that accelerated the learning process and improved the model's ability to generalise.
3. **Feature Transfer and Fine-Tuning:** The combination of freezing the base layers and fine-tuning the custom layers enabled the model to adapt the general features learned from ImageNet to the specific characteristics of CIFAR-10. The GlobalAveragePooling2D layer effectively summarised the learned features, and the custom dense layers allowed for specialisation to the CIFAR-10 dataset. This approach ensured that the most relevant features were used, contributing to better classification performance compared to the custom CNN, which had to learn all features from scratch.
4. **Robust Feature Extraction:** The ResNet50 model's use of multiple residual blocks made it highly effective at capturing both low-level and high-level features. Each residual block learns different levels of abstraction, allowing the model to capture intricate patterns in the input data. The deeper architecture and residual learning approach provided a more comprehensive understanding of the CIFAR-10 images, which resulted in improved accuracy, precision, recall, and F1 score.
5. **Regularization and Learning Rate Scheduling:** The ResNet50 model used effective regularisation techniques, such as dropout and learning rate scheduling, to enhance generalisation. The ReduceLROnPlateau callback allowed the learning rate to adapt based on model performance, enabling the model to make finer adjustments during training. This helped the ResNet50 model converge more effectively and avoid overfitting, which is often a challenge with deep architectures.

Overall, the combination of a deep architecture, pre-trained weights from ImageNet, robust feature extraction, and effective fine-tuning allowed ResNet50 to outperform the custom CNN in all evaluation metrics. The custom CNN, despite its well-thought-out design, lacked the benefit of pre-training on a large dataset and the depth necessary to extract the highly abstract features that ResNet50 could capture.

6. Insights Gained

- **Data Augmentation Improves Generalization:** Applying data augmentation helped the custom CNN model reduce overfitting and improved overall performance.
- **Transfer Learning Benefits:** Using a pre-trained ResNet50 significantly improved model performance, demonstrating the power of transfer learning when labelled data is limited.
- **Model complexity vs. Performance**: While the hybrid model performed slightly better, it also increased complexity. For some applications, the simpler standalone CNN might be preferred if the marginal gain in accuracy is not critical.

- **Class similarity**: The models struggled most with distinguishing between visually similar classes, e.g. cats and dogs. Future work could focus on developing techniques to better differentiate these challenging cases.
- **Fine-tuning Challenges**: Fine-tuning pre-trained models requires careful adjustment of hyperparameters and possibly additional training data, which may not always result in better performance for simpler datasets.

7. Future Work

- **Exploring Other Architectures:** Investigating other state-of-the-art architectures such as EfficientNet or Vision Transformers for CIFAR-10 classification.
- **Hyperparameter Optimization:** Conducting a more thorough hyperparameter search to improve the performance of the custom CNN.
- **Ensemble methods**: Exploring ensemble techniques combining multiple CNN architectures or other classifiers could potentially boost performance further.