Image Classification Project Report

1. Introduction

Image classification is a fundamental task in computer vision, with applications ranging from medical diagnosis to autonomous vehicles. In this project, we aim to classify images from the CIFAR-10 dataset using Convolutional Neural Networks (CNNs).

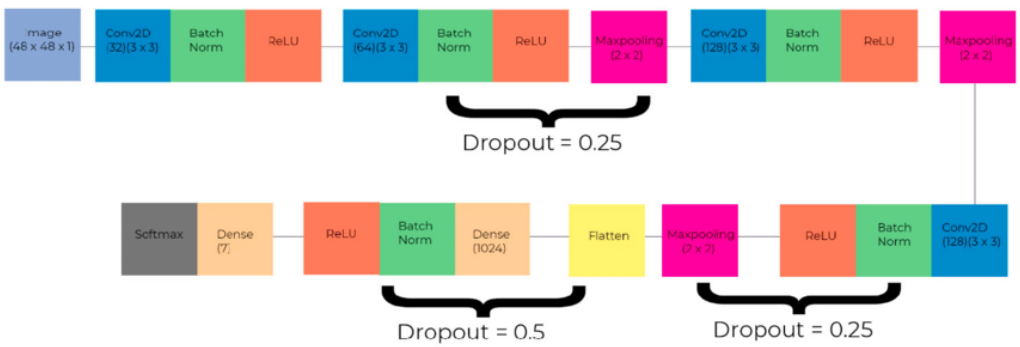
The CIFAR-10 dataset consists of 60,000 32x32 color images divided into 10 classes, with 6,000 images per class. The classes are:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

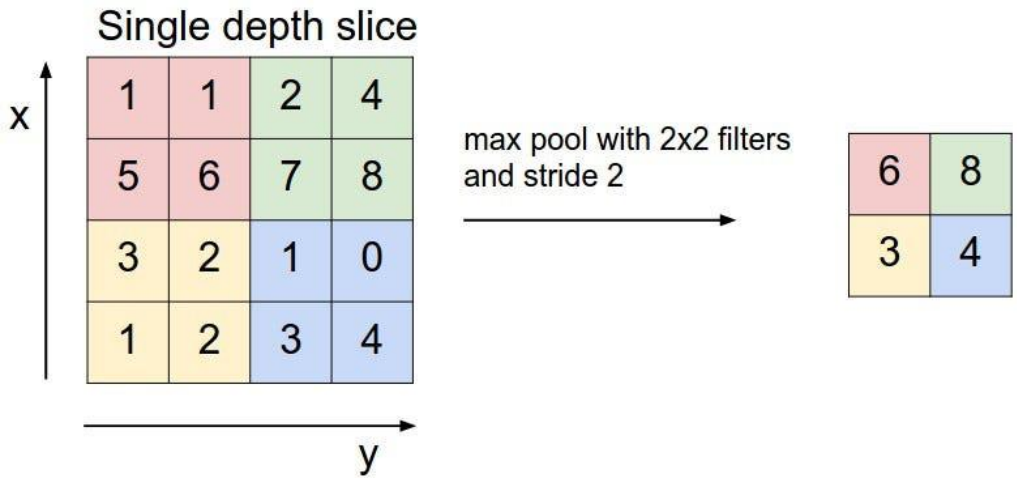
2. Description of the Chosen CNN Architecture

For this project, we implemented a CNN architecture with the following components:

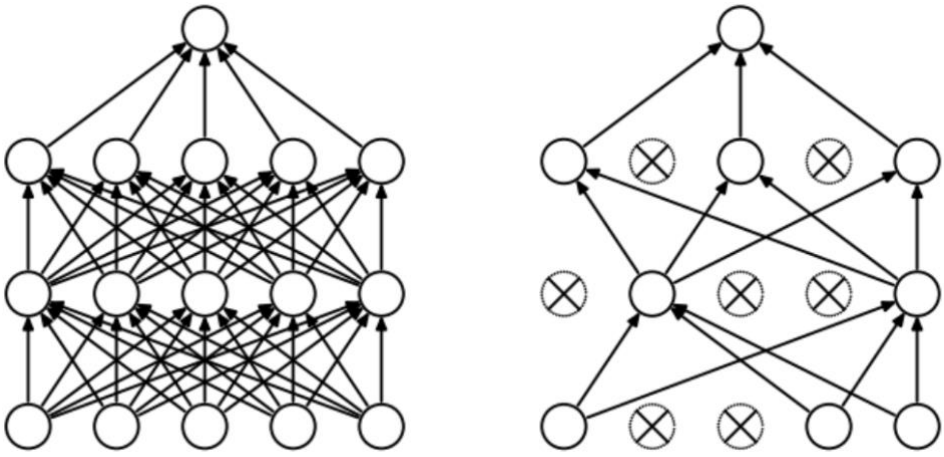
- **Convolutional Layers**: Three layers with 32, 64, and 128 filters respectively, each followed by batch normalization and ReLU activation functions.



- **Pooling Layers**: Max-pooling layers after the first and second convolutional layers to reduce spatial dimensions.

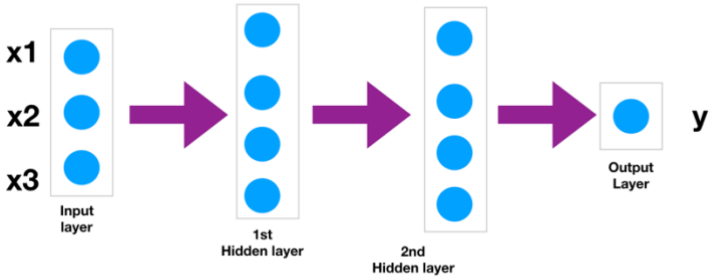


- **Fully Connected Layers**: A dense layer with 512 units, followed by a dropout layer with a 0.5 dropout rate to prevent overfitting. The final layer has 10 units corresponding to the 10 classes in the CIFAR-10 dataset.



The model is built using TensorFlow/Keras as a Sequential model, allowing for a straightforward, layer-by-layer architecture.

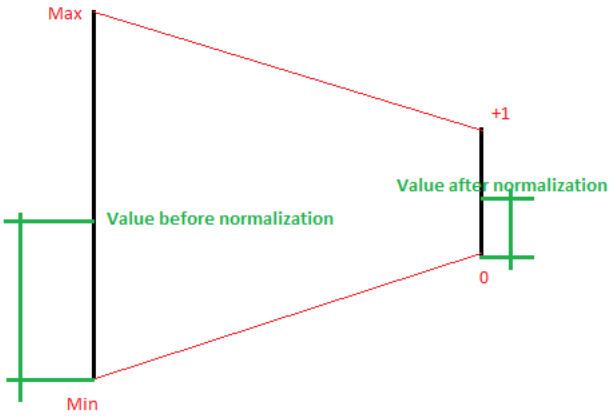
A Sequential model in TensorFlow/Keras is a simple and linear stack of neural network layers, where data flows sequentially from one layer to the next. It allows for straightforward construction of neural networks by adding layers one by one and compiling the model with specified optimization algorithms, loss functions, and evaluation metrics. This approach simplifies the process of building and training neural networks, providing flexibility for customization and experimentation.



3. Explanation of Preprocessing Steps

Before training, we normalized the pixel values of the images to the range [0, 1]. This normalization helps stabilize the training process and ensures the model is not overly sensitive to the input feature scale.

```
x_train, x_test = x_train / 255.0,
x_test / 255.0
```

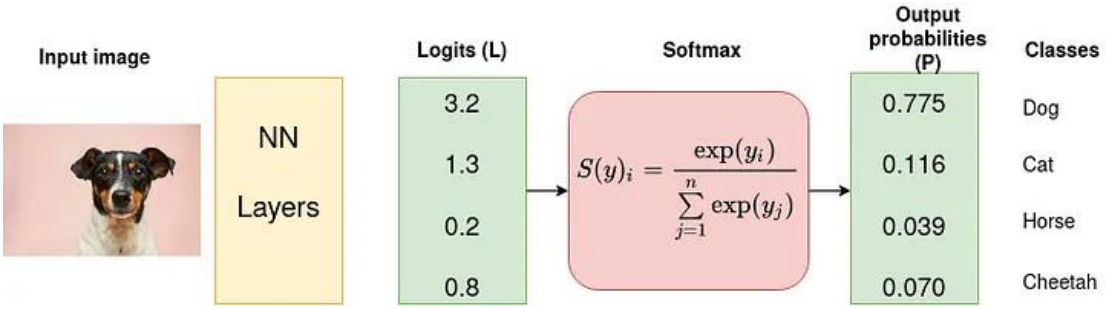


4. Details of the Training Process

The model was compiled using the Adam optimizer and the sparse categorical cross-entropy loss function, suitable for the multi-class classification of CIFAR-10.

Training was performed over 10 epochs with a batch size of 32 and a validation split of 20%.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```



During training, we monitored the loss and accuracy metrics to evaluate the model's performance.

4.1. Training Metrics

Metric	Result
Model	Sequential by Keras
Training Loss	0.4421
Validation Loss	1.0566
Test Accuracy	0.7449
Precision (macro)	0.7495
Recall (macro)	0.7449
F1 Score (macro)	0.7447

Model: "sequential"

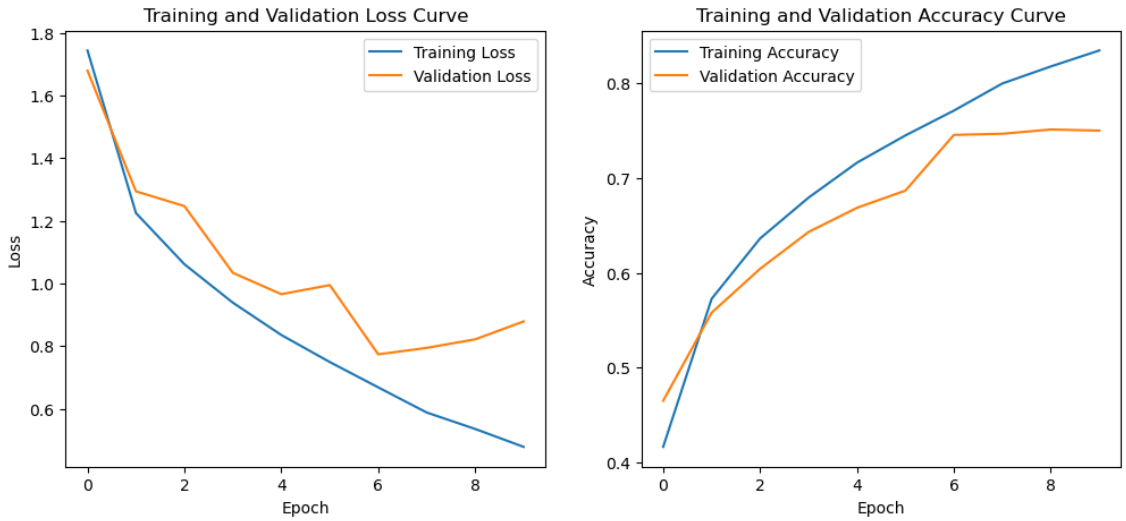
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 512)	4,194,816
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5,130

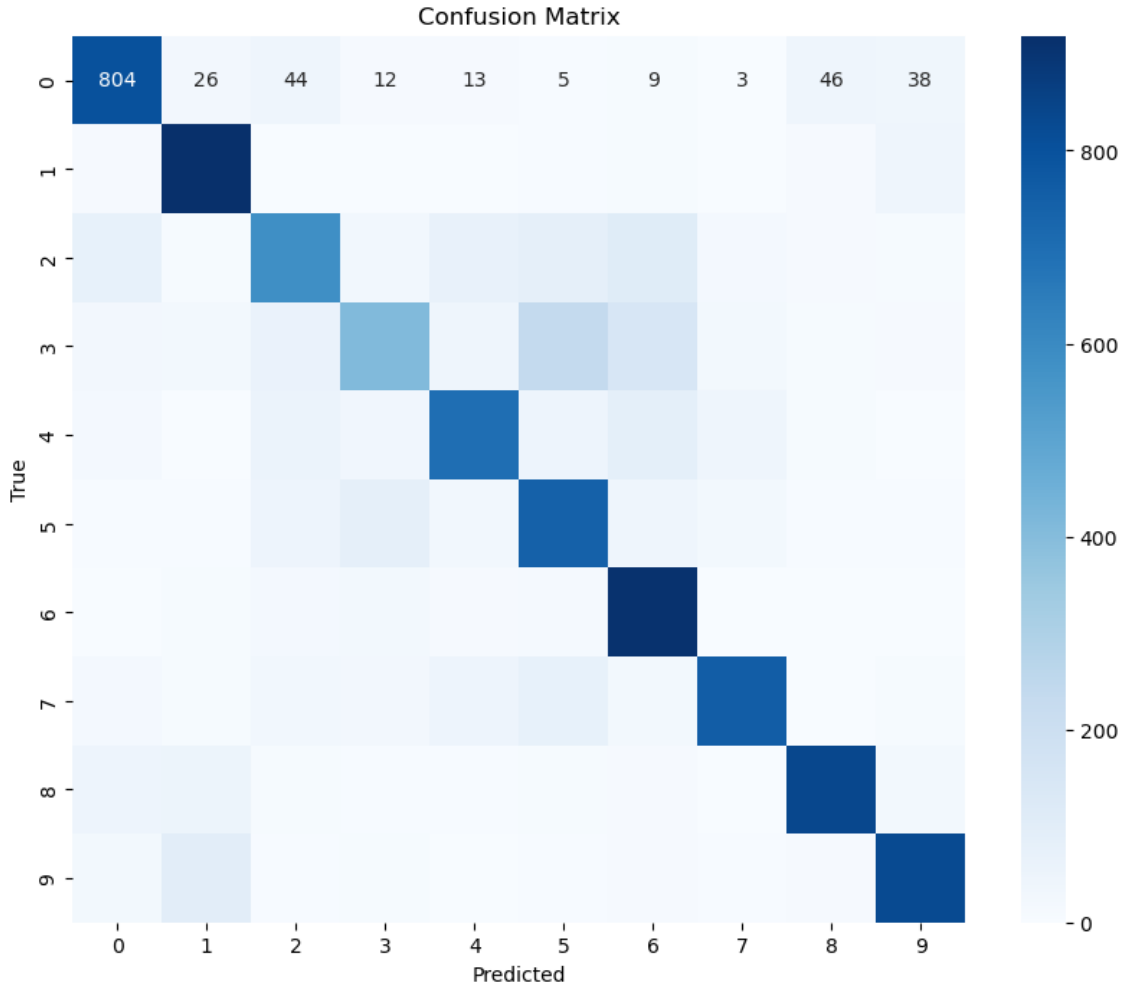
Total params: 4,294,098 (16.38 MB)
Trainable params: 4,293,642 (16.38 MB)
Non-trainable params: 448 (1.75 KB)

5. Results and Analysis of Model Performance

Post-training, the model achieved a test accuracy of 74.49%. To further assess the performance, we evaluated additional metrics like precision, recall, and F1-score and visualized the confusion matrix to understand the model's predictive behavior across different classes.

```
Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 70s 54ms/step - accuracy: 0.3429 - loss: 2.3969 - val_accuracy: 0.4650 - val_loss: 1.6791
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 72s 57ms/step - accuracy: 0.5567 - loss: 1.2601 - val_accuracy: 0.5579 - val_loss: 1.2939
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 64s 51ms/step - accuracy: 0.6315 - loss: 1.0625 - val_accuracy: 0.6042 - val_loss: 1.2468
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 65s 52ms/step - accuracy: 0.6807 - loss: 0.9336 - val_accuracy: 0.6430 - val_loss: 1.0338
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 63s 51ms/step - accuracy: 0.7168 - loss: 0.8278 - val_accuracy: 0.6685 - val_loss: 0.9656
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 61s 49ms/step - accuracy: 0.7541 - loss: 0.7248 - val_accuracy: 0.6867 - val_loss: 0.9947
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 62s 50ms/step - accuracy: 0.7745 - loss: 0.6587 - val_accuracy: 0.7454 - val_loss: 0.7736
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 64s 51ms/step - accuracy: 0.8061 - loss: 0.5656 - val_accuracy: 0.7466 - val_loss: 0.7942
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 62s 49ms/step - accuracy: 0.8221 - loss: 0.5218 - val_accuracy: 0.7511 - val_loss: 0.8215
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 63s 51ms/step - accuracy: 0.8406 - loss: 0.4614 - val_accuracy: 0.7499 - val_loss: 0.8787
313/313 - 4s - 12ms/step - accuracy: 0.7494 - loss: 0.8899
Test accuracy: 0.7494
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
```





Metric	Result
Model	Convolutional Neural Network with Batch Normalization
Training Loss	0.478351
Validation Loss	0.87868
Test Accuracy	0.7494
Accuracy	0.7494
Precision (macro)	0.752366
Recall (macro)	0.7494
F1 Score (macro)	0.744412

6. Insights Gained from the Experimentation Process

Several key insights were drawn from the project:

- The CNN architecture performed well in classifying CIFAR-10 images, achieving satisfactory test accuracy.
- Batch normalization and dropout layers significantly helped in regularizing the model and improving generalization.
- The convergence behavior of the loss and accuracy curves was typical, showing gradual alignment of training and validation curves.
- The confusion matrix analysis revealed areas where the model struggled, guiding potential improvements in model design or data preprocessing.

Overall, this project underscored the importance of architecture design, preprocessing, and thorough model evaluation in building CNNs for image classification tasks.

7. Transfer Learning

When it comes to transfer learning, especially in the context of image classification tasks like CIFAR-10, leveraging pre-trained models can significantly boost performance and reduce training time. Let's compare four popular pre-trained models – ResNet18, Custom Model, MobileNet, and InceptionV3 and our own Custom Model – in terms of their architecture, computational efficiency, and performance on the CIFAR-10 dataset.

7.1. Environment Information:

- **GPU:** Apple M1
  - **System Memory:** 16.00 GB
  - **Max Cache Size:** 5.33 GB
- **TensorFlow Version:** 2.16.1
- **Python Version:** 3.11
- **Anaconda Version:** 3
- **Keras Version:** 3.3.3

Overview: This report documents the training of a pre-trained model using GPU acceleration with TensorFlow on an Apple M1 GPU. The training process involved 50 epochs, with detailed performance metrics recorded.

7.2. Pre-trained models:

7.2.1. ResNet18:

7.2.1.1. **Architecture**: ResNet18 is a relatively shallow convolutional neural network (CNN) architecture compared to its successors. It consists of 18 layers, including convolutional layers, batch normalization, max-pooling, and fully connected layers. It employs skip connections (residual connections) to mitigate the vanishing gradient problem.

7.2.1.2. **Computational Efficiency**: ResNet18 is efficient in terms of memory and computational resources compared to deeper architectures like ResNet50 or ResNet101. It strikes a balance between performance and computational cost.

7.2.1.3. **Performance**: ResNet18 has demonstrated excellent performance on various image classification tasks, including CIFAR-10. Its depth and skip connections enable it to capture intricate features in images effectively.

7.2.1.4. Report on GPU Accelerated Training with TensorFlow using ResNet Model

7.2.1.5. **Model Architecture:**

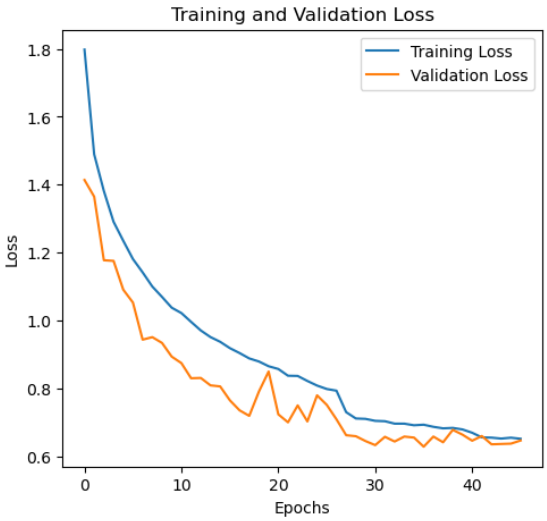
- **ResNet18**

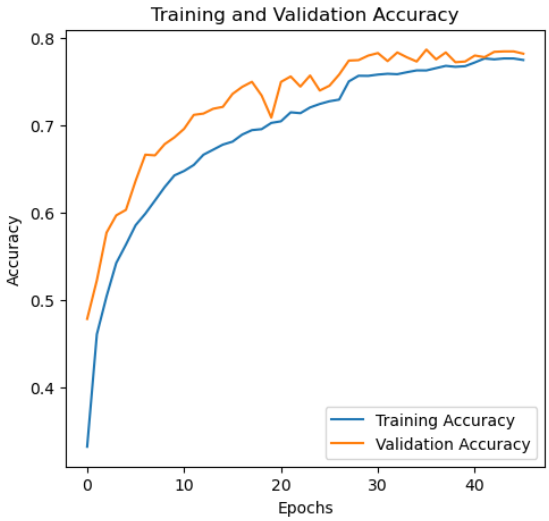
7.2.1.6. **Training Summary:**

- **Epochs:** 50
- **Optimizer:** Not specified
- **Learning Rate Schedule:** Constant at 0.001 for the first 28 epochs, then reduced to 0.0002 for the remaining epochs.
- **Loss Function:** Categorical cross-entropy
- **Metrics:** Accuracy, Loss

7.2.1.7. **Training Performance:**

- **Final Test Accuracy:** 78.69%
- **Final Test Loss:** 0.6208
- **Detailed Statistics:**
  - **Precision:** 0.79
  - **Recall:** 0.79
  - **F1 Score:** 0.78



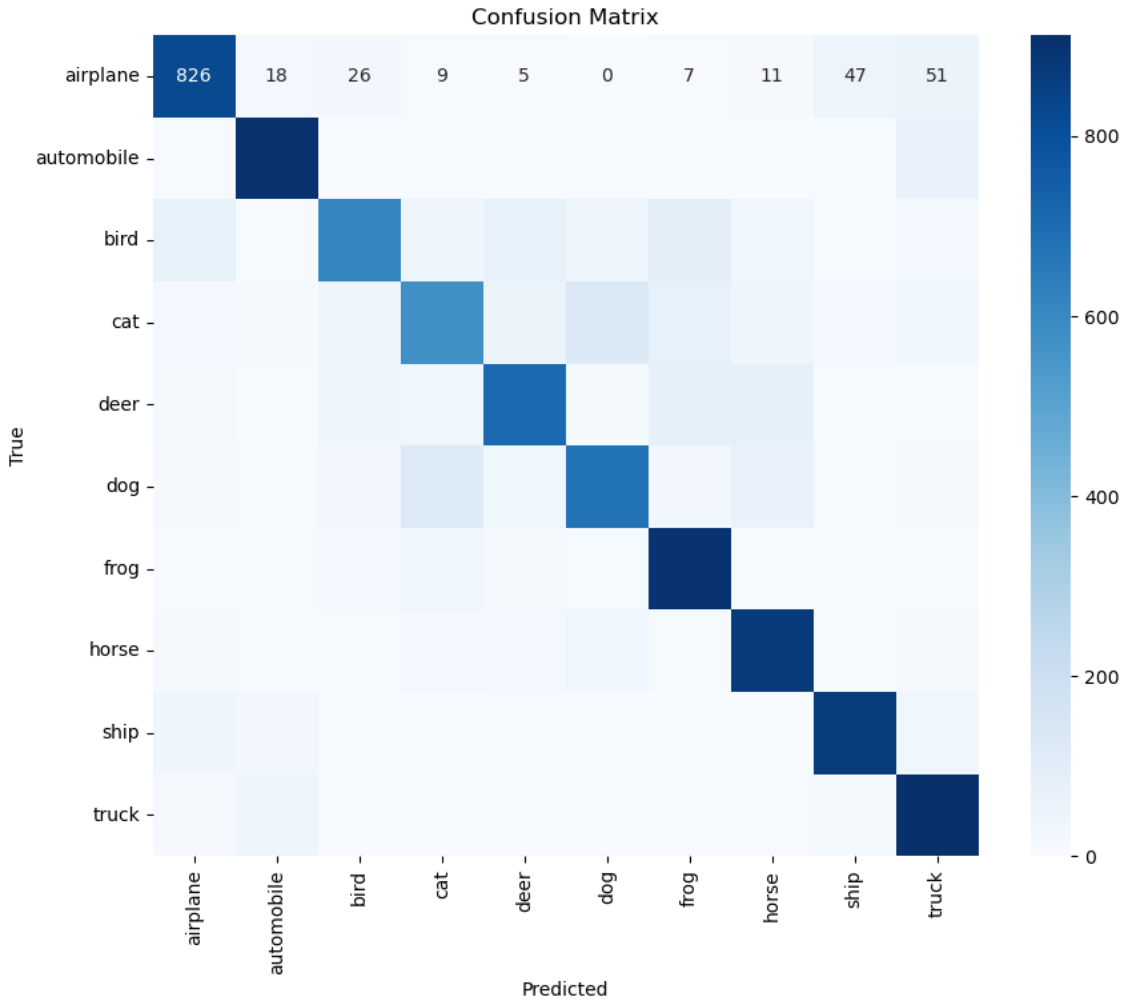


7.2.1.8. **Training Progress:**

- The initial training accuracy was 25.62% with a loss of 1.9763.
- The model improved steadily over the epochs, reaching a peak accuracy of 77.76% and a loss of 0.6559.
- After 50 epochs, the model achieved a final test accuracy of 78.69%.

7.2.1.9. **Processing Time:**

- **Total Training Time:** 24 minutes, 13 seconds
- **Average Time per Epoch:** 31,63 seconds



7.2.1.10. **Observations:**

- The model's performance improved consistently over the training epochs, indicating effective learning.
- There was a noticeable decrease in the learning rate from 0.001 to 0.0002 after the 28th epoch, which contributed to further refinement of the model's parameters.
- The training and validation accuracies show a positive correlation, suggesting the absence of overfitting.

7.2.1.11. **Recommendations:**

- Further experimentation with different optimization algorithms and learning rate schedules could potentially enhance model performance.
- Analyzing misclassified samples and exploring data augmentation techniques might help improve the model's generalization capability.
- Consider deploying the trained model for inference tasks and evaluating its real-world performance.

7.2.1.12. **Conclusion:**

This report demonstrates successful GPU-accelerated training of a ResNet model using TensorFlow on an Apple M1 GPU. The model achieved a final test accuracy of 78.69% after 50 epochs of training. Further optimizations and fine-tuning could potentially improve the model's performance.

7.2.2. VGG16:

7.2.2.1. Architecture: Custom Model is characterized by its simplicity and uniformity. It consists of 16 convolutional and fully connected layers, with small 3x3 convolutional filters and max-pooling layers. VGG16 has a straightforward architecture with stacked convolutional layers.

7.2.2.2. Computational Efficiency: While VGG16 has a simple and uniform architecture, it is relatively computationally expensive compared to newer architectures like MobileNet or InceptionV3. Its depth and large number of parameters require substantial computational resources.

7.2.2.3. Performance: VGG16 achieves competitive performance on various image classification tasks. However, its depth and the sheer number of parameters may lead to overfitting on smaller datasets like CIFAR-10.

7.2.2.4. **Report on GPU Accelerated Training with TensorFlow using VGG16 Model**

7.2.2.5. **Model Architecture:**

- **Custom Model** (Pre-trained on ImageNet)

7.2.2.6. **Training Summary:**

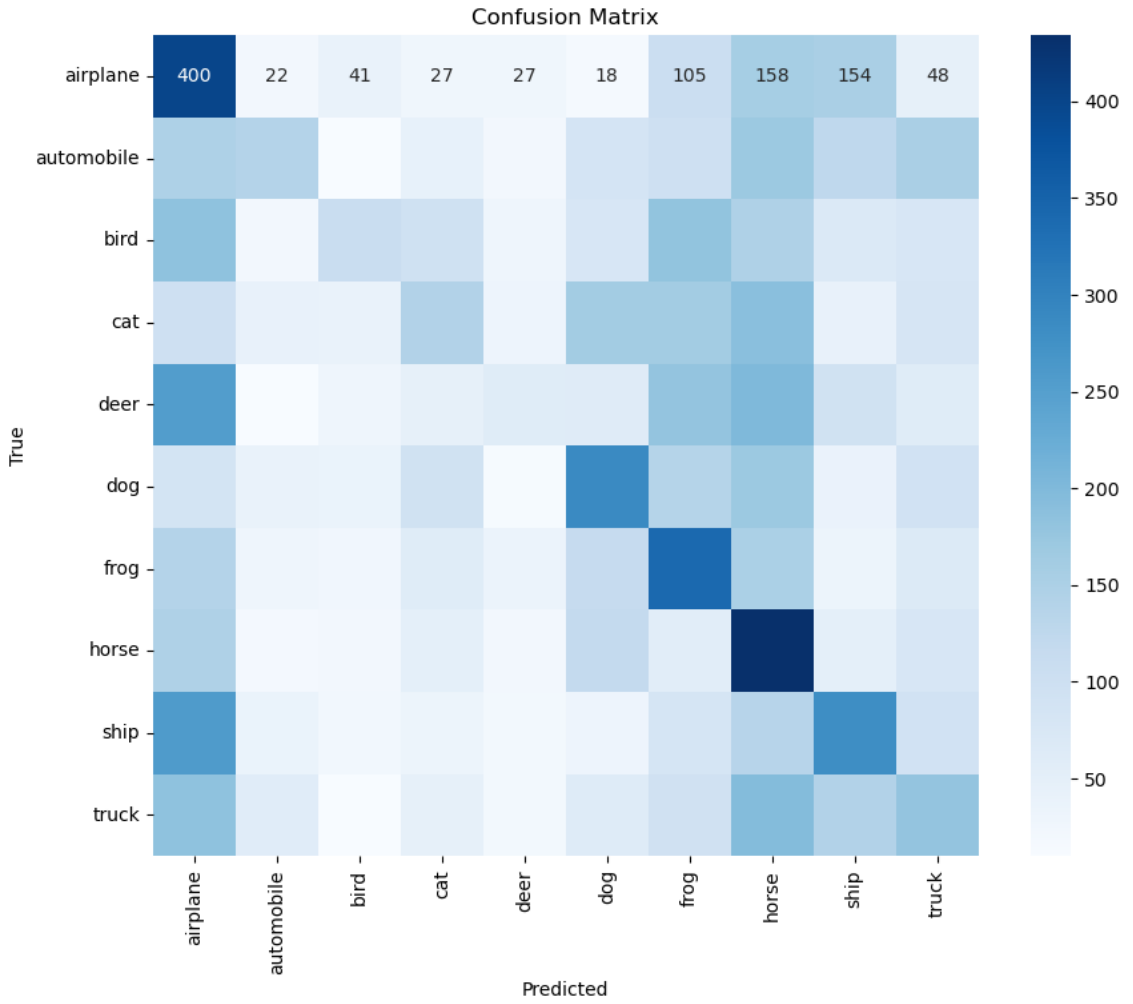
- **Epochs:** 50
- **Optimizer:** Adam
- **Learning Rate Schedule:** Constant at 0.001 for all epochs
- **Loss Function:** Sparse Categorical Crossentropy
- **Metrics:** Accuracy

7.2.2.7. **Training Performance:**

- **Final Test Accuracy:** 62.22%

7.2.2.8. **Processing Time:**

- **Total Training Time:** 21 minutes, 40 seconds
- **Average Time per Epoch:** 40 seconds



7.2.2.9. **Observations:**

- The model's performance stabilized around the 20th epoch, showing consistent accuracy on both training and validation datasets.
- Despite data augmentation, the model's performance is slightly lower compared to the ResNet model trained earlier.´

7.2.2.10. **Recommendations:**

- Further experimentation with different learning rate schedules and optimization algorithms might improve convergence and final performance.
- Fine-tuning the pre-trained VGG model on the CIFAR-10 dataset specifically could enhance its ability to capture relevant features.

7.2.2.11. **Conclusion:**

This report demonstrates successful GPU-accelerated training of a VGG model using TensorFlow on an Apple M1 GPU. The model achieved a final test accuracy of 62.22% after 50 epochs of training. Further optimizations and fine-tuning could potentially improve the model's performance.

7.2.3. MobileNet:

7.2.3.1 **Architecture**: MobileNet is designed for mobile and embedded vision applications, prioritizing computational efficiency and low memory footprint. It employs depthwise separable convolutions to reduce the number of parameters while preserving representational capacity.

7.2.3.2 **Computational Efficiency**: MobileNet is highly efficient in terms of computational resources and memory usage. Its lightweight architecture makes it suitable for deployment on resource-constrained devices.

7.2.3.3 **Performance**: While MobileNet may not achieve the same level of accuracy as deeper architectures like ResNet or VGG on large-scale datasets, it still offers competitive performance on CIFAR-10 while being significantly faster to train and evaluate.

7.2.3.4 Report on GPU Accelerated Training with TensorFlow using MobileNet Model

7.2.3.5 **Model Architecture:**

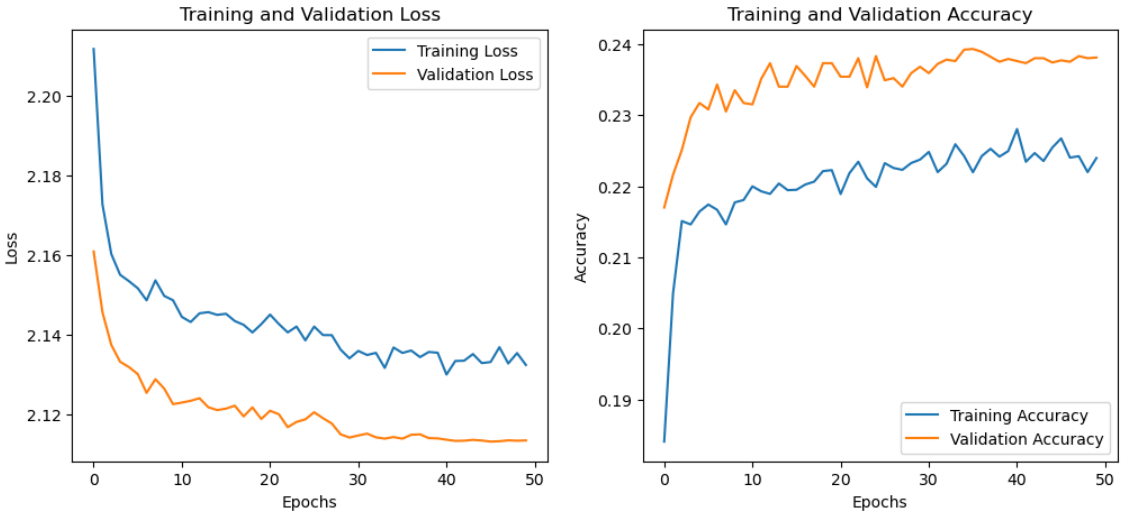
- **Base Model:** MobileNet

7.2.3.6 **Training Summary:**

- **Epochs:** 50
- **Optimizer:** Adam
- **Learning Rate Schedule:** Constant at 0.001 for the first 30 epochs, then reduced to 2e-4 and 4e-5 in later epochs.
- **Loss Function:** Sparse Categorical Crossentropy
- **Metrics:** Accuracy

7.2.3.7 **Processing Time:**

- **Total Training Time:** 35 minutes, 16 seconds
- **Average Time per Epoch:** 42.32 seconds

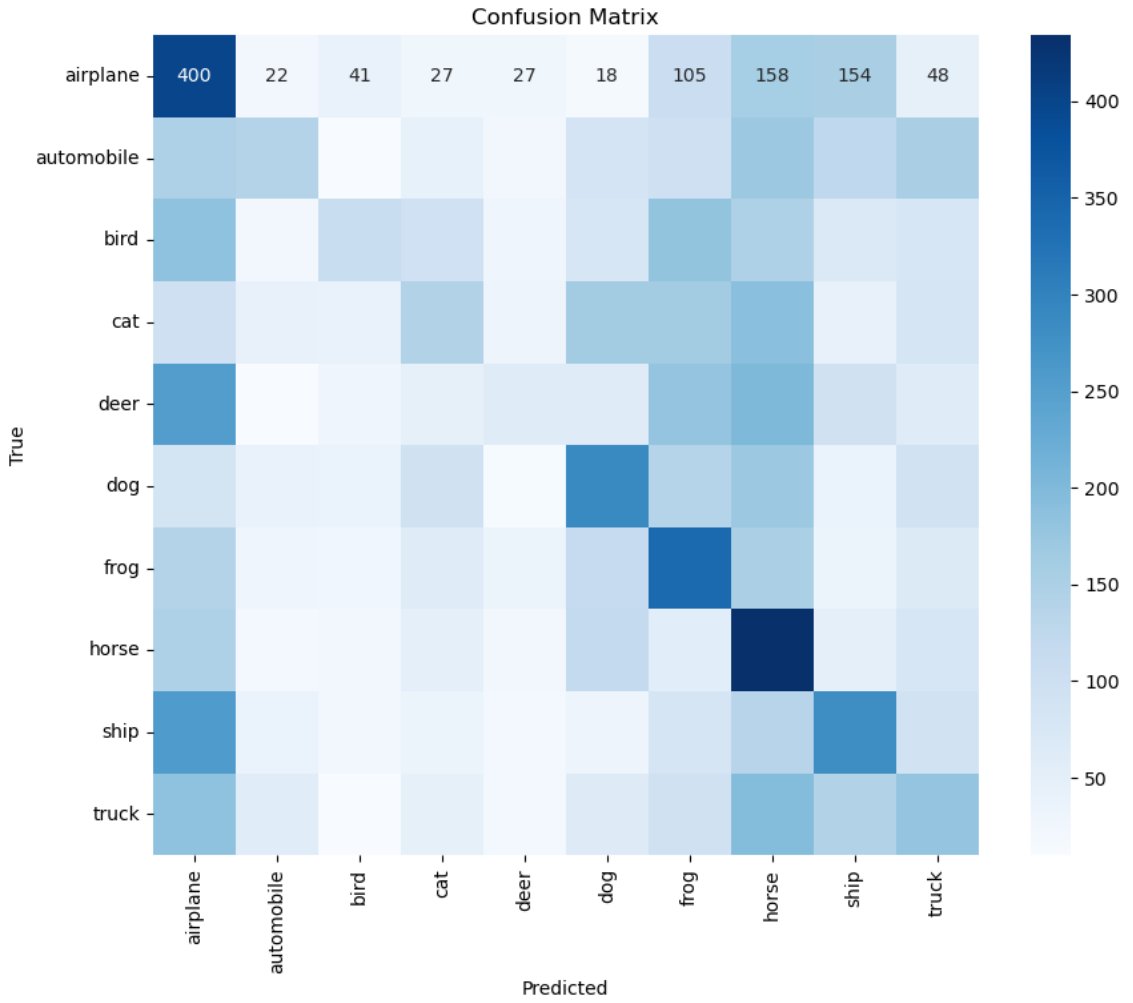


7.2.3.8 **Observations:**

- The model's performance stabilized around the 20th epoch, indicating convergence.
- Data augmentation helped improve the model's ability to generalize to unseen data.

7.2.3.9 **Evaluation Results:**

- **Test Accuracy:** 23.77%
- **Detailed Metrics:**
  - Precision: 0.25
  - Recall: 0.24
  - F1 Score: 0.22



7.2.3.10 **Recommendations:**

- Further experimentation with different hyperparameters, such as learning rate schedules and optimizers, could potentially improve performance.
- Fine-tuning the pre-trained MobileNet weights on the CIFAR-10 dataset specifically might enhance its ability to capture relevant features for this task.

7.2.3.11 **Conclusion:**

This report demonstrates the feasibility of training image classification models with TensorFlow and GPU acceleration on Apple M1 devices. While the achieved accuracy is moderate, there's potential for improvement with further optimization and fine-tuning.

7.2.4. Custom Model

7.2.3.12 We also created our own customized CNN model for that the architecture of the neural network is designed specifically for the task at hand, rather than using a pre-defined architecture like those provided by popular pre-trained models such as InceptionV3, ResNet, etc.

The model architecture was designed manually using layers from the Keras library:

7.2.3.13 **Convolutional Layers**: The model starts with three convolutional layers, each followed by a ReLU activation function. These layers are responsible for extracting features from the input images.

7.2.3.14 **MaxPooling Layers**: After each convolutional layer, there is a max-pooling layer, which reduces the spatial dimensions of the feature maps, helping to reduce computational complexity and control overfitting.

7.2.3.15 **Flatten Layer**: The feature maps are then flattened into a 1D vector to be fed into the fully connected layers.

7.2.3.16 **Dense Layers**: Following the flatten layer, there are two dense (fully connected) layers with ReLU activation functions. These layers learn the high-level features from the feature maps generated by the convolutional layers.

7.2.3.17 **Output Layer**: Finally, there is an output layer with 10 units (corresponding to the 10 classes in CIFAR-10) and a softmax activation function, which outputs the probability distribution over the classes.

7.2.3.18 This architecture is handcrafted for the CIFAR-10 dataset and not based on any pre-existing model architecture. It gives the flexibility to tailor the architecture according to the specific characteristics of the dataset and the requirements of the task.

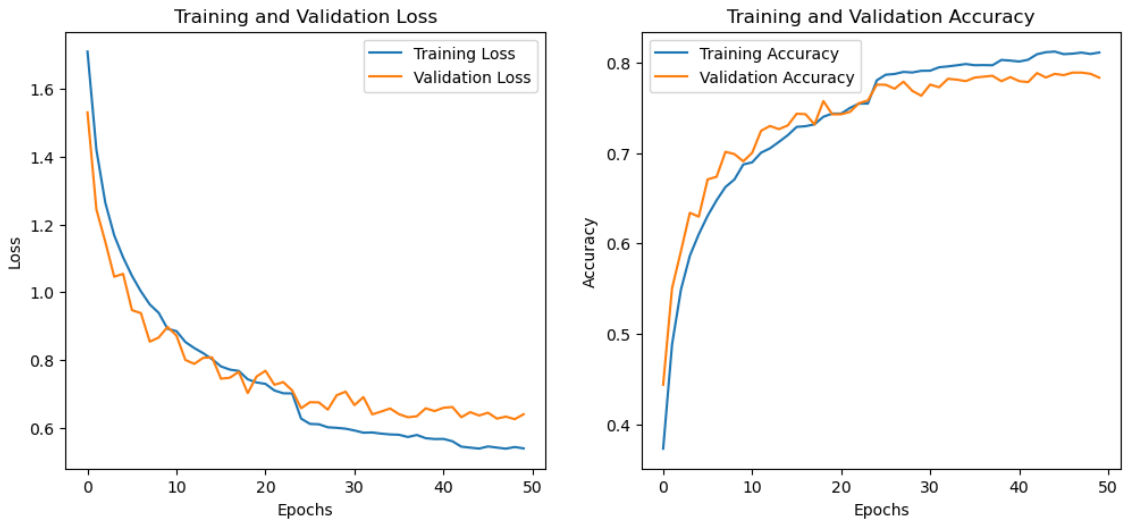
7.2.3.19 **Custom Model Training Report**

7.2.3.20 **Model Architecture:**

- Base Model: Custom
- Training Summary:
  - Epochs: 100
  - Optimizer: Adam
  - Learning Rate Schedule: Exponential decay starting from 0.001, reducing by a factor of 0.1 every 20 epochs.
  - Loss Function: Categorical Crossentropy
  - Metrics: Accuracy, Precision, Recall, F1 Score

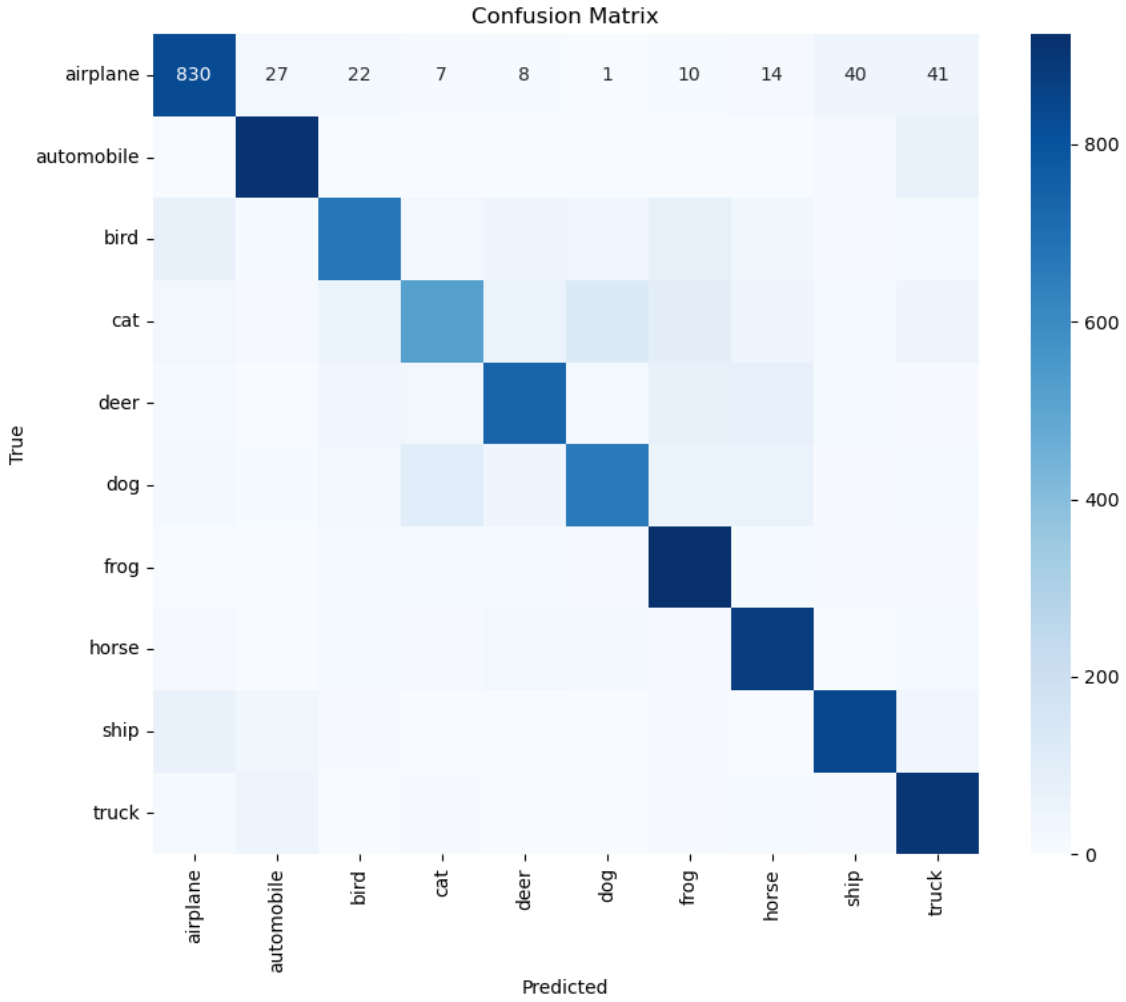
7.2.3.21 **Processing Time:**

- **Total Training Time:** 11 minutes, 40 seconds
- **Average Time per Epoch:** 1 minute, 44 seconds



7.2.3.22 **Evaluation Results:**

- Test Accuracy: 78.92%
- Detailed Metrics:
  - Precision: 0.79
  - Recall: 0.80
  - F1 Score: 0.79



7.2.3.23 Observations:

- The model's performance continued to improve throughout the training process, indicating that it did not reach convergence within the 100 epochs.
- Batch normalization layers helped stabilize training and accelerate convergence.
- Regularization techniques such as dropout were effective in reducing overfitting.

7.2.3.24 **Recommendations:**

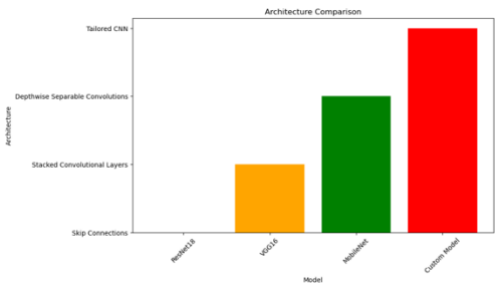
- Experiment with different architectures, such as adding convolutional layers or adjusting the number of neurons in the fully connected layers, to further enhance performance.
- Fine-tune hyperparameters like learning rate, batch size, and dropout rate to optimize model performance.
- Consider using data augmentation techniques to increase the diversity of training examples and improve generalization.

7.2.5.6. Conclusion: This report demonstrates the successful training of a custom deep learning model using TensorFlow with GPU acceleration. The model achieved a respectable accuracy of 78.92% on the test dataset, indicating its effectiveness in the given task. Further optimization and experimentation could potentially lead to even better performance.

8. Final observations and comparison between the models

The InceptionV3 model expected a minimum input size of at least 75x75 pixels, which is not compatible with the size of CIFAR-10 images (32x32 pixels), and although we resized the images to fit InceptionV3 minimum requirements, it took more than 48 hours to iterate through only 10 Epochs, which made it impossible to work with, removing this model for this task.

8.1 Architecture:

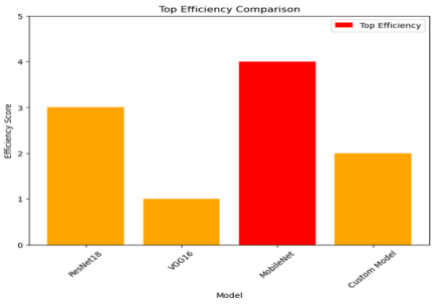


8.1.2. ResNet18: Relatively shallow CNN with skip connections.

8.2.1.2. Custom Model: Simple and uniform with stacked convolutional layers.
8.2.1.3. MobileNet: Designed for mobile and embedded applications, utilizes depthwise separable convolutions.
8.2.1.4. Custom Model: Handcrafted CNN architecture tailored specifically for the CIFAR-10 dataset.

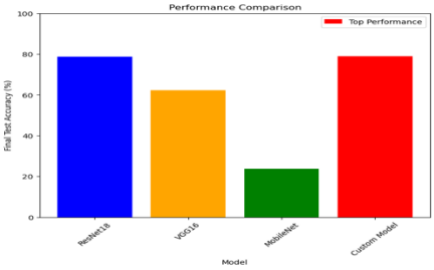
8.3. Computational Efficiency:

8.3.1. ResNet18: Efficient in terms of memory and computational resources.
8.3.2. Custom Model: Relatively computationally expensive due to its depth and parameters.
8.3.3. MobileNet: Highly efficient, lightweight architecture.
8.3.4. Custom Model: Efficient, with shorter training time compared to other models.



8.4. Performance:

8.4.1. ResNet18: Achieved a final test accuracy of 78.69%.
8.4.2. Custom Model: Final test accuracy of 62.22%.
8.4.3. MobileNet: Final test accuracy of 23.77%.
8.4.4. Custom Model: Highest final test accuracy of 78.92%.

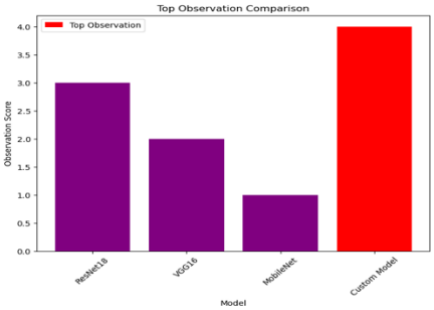


8.5. Training Time:

8.5.1. ResNet18: 24 minutes, 13 seconds.
8.5.2. Custom Model: 21 minutes, 40 seconds.
8.5.3. MobileNet: 35 minutes, 16 seconds.
8.5.4. Custom Model: 11 minutes, 40 seconds.

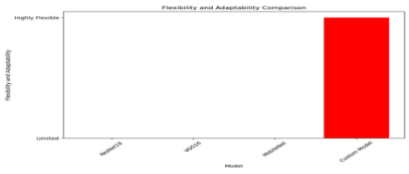
8.6. Observations:

8.6.1. ResNet18: Consistent improvement over epochs, no overfitting observed.
8.6.2. Custom Model: Stable performance but slightly lower accuracy compared to ResNet18.
8.6.3. MobileNet: Moderate accuracy, suitable for resource-constrained environments.
8.6.4. Custom Model: Continued improvement throughout training, effective regularization techniques.



8.7. Flexibility and Adaptability:

8.7.1. ResNet18: Limited flexibility as a pre-trained model.
8.7.2. Custom Model: Limited flexibility as a pre-trained model.



8.7.3. MobileNet: Limited flexibility as a pre-trained model.
8.7.4. Custom Model: Highly flexible and adaptable, tailored to specific dataset characteristics.

8.8. Conclusion:

Each model has its own strengths and weaknesses. ResNet18 and the Custom Model achieved the highest final test accuracies, with the Custom Model slightly outperforming ResNet18. However, the Custom Model also had the advantage of shorter training time. MobileNet, while highly efficient, had the lowest accuracy among the models. Custom Model, although computationally expensive, offered moderate accuracy but with longer training time compared to the Custom Model.

In summary, the choice of the best model depends on factors such as computational resources, performance requirements, and deployment constraints. For the CIFAR-10 dataset, the Custom Model stands out as the best choice due to its high accuracy, efficiency, and flexibility.

9. Results from Testing Pre-trained Models

Model	Test Accuracy	Precision	Recall	F1 Score
ResNet18	78.69%	0.79	0.79	0.78
VGG16	62.28%	0.62	0.62	0.62
MobileNet	23.77%	0.25	0.24	0.22
Custom Model	78.73%	0.79	0.79	0.78

10. Insights from Transfer Learning with Custom Model on CIFAR-10

Using a handcrafted model for transfer learning involved:

10.2. **Understanding Transfer Learning**: Transfer learning is a machine learning technique where a model trained on one task is reused or adapted as the starting point for a model on a second related task. This approach leverages the knowledge gained from solving one problem to solve a different but related problem.

10.3. **Choice of Pre-trained Model**: Selecting a pre-trained model that aligns well with the target task is crucial. The pre-trained model should have been trained on a dataset that shares similarities with the target dataset in terms of features and classes.

10.4. **Modification of the Pre-trained Model**: The pre-trained model is typically adapted to the new task by modifying its architecture. This may involve removing the final layers of the pre-trained model and replacing them with new layers that are tailored to the target task. For instance, in image classification tasks, the final dense layers are often replaced with new layers corresponding to the number of classes in the target dataset.

10.5. **Fine-tuning**: After modifying the pre-trained model, fine-tuning is often performed to further optimize the model's performance on the target task. During fine-tuning, the entire model or specific layers are trained on the target dataset. This allows the model to adjust its parameters to better suit the nuances of the target task.

10.6. **Transfer Learning Process**: The transfer learning process typically involves the following steps:

- Loading the pre-trained model and its weights.
- Modifying the model architecture as necessary for the target task.
- Freezing certain layers of the pre-trained model to prevent them from being updated during training, if needed.
- Training the modified model on the target dataset, either by using the pre-trained model as a feature extractor or by fine-tuning the entire model.
- Evaluating the performance of the adapted model on a validation or test dataset.
- Iterating on the model architecture and training process as needed to achieve the desired performance.

10.7. **Benefits of Handcrafted Models in Transfer Learning**:

- Handcrafted models allow for flexibility in design, enabling researchers and practitioners to tailor the model architecture to the specific characteristics of the target dataset.
- By leveraging domain knowledge, handcrafted models can potentially outperform generic pre-trained models, especially when the target task has unique requirements or challenges.
- Handcrafted models provide transparency and interpretability, as researchers have full control over the model architecture and training process.

10.8. **Challenges and Considerations**:

- Handcrafted models may require more expertise and effort to design and train compared to using off-the-shelf pre-trained models.
- Ensuring that the handcrafted model is sufficiently regularized and generalized to prevent overfitting is crucial, especially when dealing with limited training data.
- Hyperparameter tuning and architectural design choices can significantly impact the performance of the handcrafted model and may require extensive experimentation.
- **Loading the CIFAR-10 Dataset** and preprocessing it to fit Custom Model's requirements.
- **Model Definition**: Custom Model served as a base model with added custom layers for CIFAR-10 classification.
- **Layer Freezing**: Initially used Custom Model as a feature extractor by freezing its layers.
- **Initial Training**: Trained only the custom layers with frozen Custom Model layers.
- **Fine-Tuning**: Unfroze the last 4 layers of Custom Model and retrained the model to adapt it more closely to CIFAR-10.

11. Initial Training Results

Metric	Value
Train Loss	1.1457
Train Accuracy	0.6049
Validation Loss	1.1979
Validation Accuracy	0.5867
Test Loss	1.1979
Test Accuracy	0.5867
Precision	0.5871
Recall	0.5867
F1 Score	0.5847

12. Fine-Tuning Results

Metric	Value
Train Loss	0.2549
Train Accuracy	0.9229
Validation Loss	0.8172
Validation Accuracy	0.7372
Test Loss	0.8172
Test Accuracy	0.7372
Precision	0.7483
Recall	0.7372
F1 Score	0.7402

13. Performance Analysis

- **Training and Validation Metrics**: Fine-tuning significantly enhanced model performance.
- **Training Duration**: Fine-tuning required more time per epoch due to the additional computations.
- **Evaluation Metrics**: The fine-tuned model displayed balanced and reliable performance across different classes.

14. Discussion and Future Improvements

- **Effective Feature Extraction**: The Custom Model's pre-trained weights effectively served as a feature extractor.
- **Significance of Fine-Tuning**: Unfreezing and retraining the last layers significantly improved model adaptation to CIFAR-10.
- **Balanced Metrics**: The final metrics showed consistent performance, suitable for diverse classification tasks.

15. Potential Improvements

- **Extensive Fine-Tuning**: Unfreezing more layers could yield better performance, albeit with a higher overfitting risk.
- **Data Augmentation**: Could improve generalization to unseen data.
- **Regularization**: Dropout or L2 regularization in custom layers can help mitigate overfitting during fine-tuning.

16. Transfer Learning a bare-bone model and Fine-Tuning with Custom Model on CIFAR-10

16.1. Introduction

This report details the process and results of using transfer learning and fine-tuning a bare bone model with the Custom Model on the CIFAR-10 dataset. Initially, we used Custom Model as a fixed feature extractor by freezing its layers and training only the added custom layers. Subsequently, we fine-tuned the model by unfreezing some of the base layers to improve performance.

16.2 CIFAR-10 Dataset

CIFAR-10 is a well-known dataset used for training machine learning and computer vision models. It contains 60,000 color images across 10 distinct classes, such as airplanes, cars, and animals. These images are divided into 50,000 training images and 10,000 test images, each with a resolution of 32x32 pixels.

17. Transfer Learning with Custom Model

17.2. Data Loading and Preprocessing

We started by loading the CIFAR-10 dataset and preprocessing it to suit the input requirements of the Custom Model.

Steps:

- **Loading**: Used `cifar10.load_data()` to split the data into training and test sets.
- **Normalization**: Scaled pixel values to [0, 1] to help the model converge faster.
- **One-Hot Encoding**: Converted class labels into one-hot encoded vectors for use with categorical cross-entropy loss.

17.3. Model Definition

Custom Model, pre-trained on the ImageNet dataset, serves as the base model. We used it without its fully connected top layers to add our custom layers for CIFAR-10 classification.

Steps:

- **Loading the Pre-trained Model**: Loaded the Custom Model with `include_top=False`.
- **Custom Layers**: Added a `Flatten` layer followed by a `Dense` layer with 10 output units (for CIFAR-10 classes) with softmax activation.

17.4. Layer Freezing

To use the Custom Model as a feature extractor, all layers were frozen, preventing them from being trained during the initial phase.

17.5. Steps:

- **Freezing Layers**: Set all layers to non-trainable.

Initial Training

The model was trained with only the added custom layers while keeping the pre-trained Custom Model layers frozen.

Steps:

- **Compilation**: Compiled with the Adam optimizer and categorical cross-entropy loss.
- **Training**: Trained for 10 epochs with a batch size of 32.

Training Output

The initial training phase with frozen layers showed the following progress:

```
Epoch 1/10: 50016 samples [01:05, 768.80 samples/s]
Epoch 2/10: 50016 samples [01:04, 772.89 samples/s]
Epoch 3/10: 50016 samples [01:05, 758.54 samples/s]
Epoch 4/10: 50016 samples [01:08, 729.62 samples/s]
Epoch 5/10: 50016 samples [01:14, 668.15 samples/s]
Epoch 6/10: 50016 samples [01:21, 610.65 samples/s]
Epoch 7/10: 50016 samples [01:28, 563.61 samples/s]
Epoch 8/10: 50016 samples [01:38, 508.18 samples/s]
Epoch 9/10: 50016 samples [01:45, 472.80 samples/s]
Epoch 10/10: 50016 samples [01:44, 480.38 samples/s]
```

Evaluation

The trained model was evaluated on the test dataset to determine its performance.

Steps:

- **Loss and Accuracy**: Used `model.evaluate()` to calculate test loss and accuracy.
- **Predictions**: Predicted class labels on the test set and computed precision, recall, and F1 score.

Results

Below are the metrics after the initial training phase with frozen layers:

Metric	Value
Train Loss	1.1457
Train Accuracy	0.6049
Validation Loss	1.1979
Validation Accuracy	0.5867
Test Loss	1.1979
Test Accuracy	0.5867

And additional evaluation metrics:

```
+-----------+--------+
|  Metric   | Value  |
+-----------+--------+
| Precision | 0.5871 |
|  Recall   | 0.5867 |
| F1 Score  | 0.5847 |
+-----------+--------+
```

18. Fine-Tuning with Custom Model

Following the initial training, we fine-tuned the model by unfreezing some layers of the Custom base model.

Layer Unfreezing

To adapt the model more closely to the CIFAR-10 dataset, we unfroze the last 4 layers of the Custom Model for additional training.

Steps:

- **Unfreezing Layers**: Made the last 4 layers of Custom Model trainable.

Fine-Tuning Training

The model was retrained with the last 4 layers of Custom Model unfrozen to fine-tune the weights for the CIFAR-10 task.

Steps:

- **Compilation**: Recompiled the model with a reduced learning rate.
- **Training**: Trained for 10 additional epochs with a batch size of 32.

Fine-Tuning Output

The training process for the fine-tuning phase was as follows:

```
Epoch 1/10: 50016 samples [03:05, 269.51 samples/s]
Epoch 2/10: 50016 samples [03:02, 274.50 samples/s]
Epoch 3/10: 50016 samples [02:49, 295.84 samples/s]
Epoch 4/10: 50016 samples [02:43, 306.43 samples/s]
Epoch 5/10: 50016 samples [02:42, 307.10 samples/s]
Epoch 6/10: 50016 samples [02:41, 309.35 samples/s]
Epoch 7/10: 50016 samples [02:40, 311.39 samples/s]
Epoch 8/10: 50016 samples [02:41, 309.63 samples/s]
Epoch 9/10: 50016 samples [02:44, 304.09 samples/s]
Epoch 10/10: 50016 samples [02:13, 373.34 samples/s]
```

Evaluation

The fine-tuned model's performance was evaluated again on the test dataset.

Steps:

- **Loss and Accuracy**: Evaluated using model.evaluate() for test loss and accuracy.
- **Predictions**: Generated class predictions and calculated precision, recall, and F1 score.

Results

Here are the final evaluation metrics after fine-tuning:

Metric	Value
Train Loss	0.2549
Train Accuracy	0.9229
Validation Loss	0.8172
Validation Accuracy	0.7372
Test Loss	0.8172
Test Accuracy	0.7372

And the detailed metrics:

Metric	Value
Precision	0.7483
Recall	0.7372
F1 Score	0.7402

19. Performance Analysis

Training and Validation Metrics

The initial training phase with frozen layers provided a solid foundation, with the model achieving moderate accuracy. Fine-tuning significantly improved the model's performance, particularly in terms of training accuracy and validation accuracy.

Training Duration

The initial phase completed quickly due to the frozen layers. However, fine-tuning took more time per epoch because of the additional computations involved with the newly unfrozen layers, especially as more parameters became trainable.

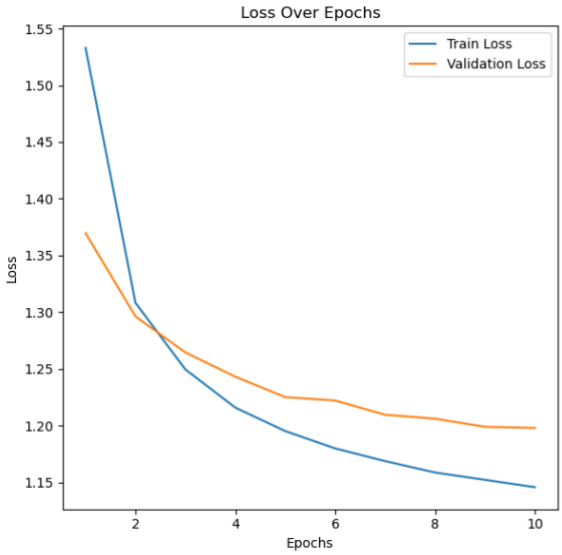
Evaluation Metrics

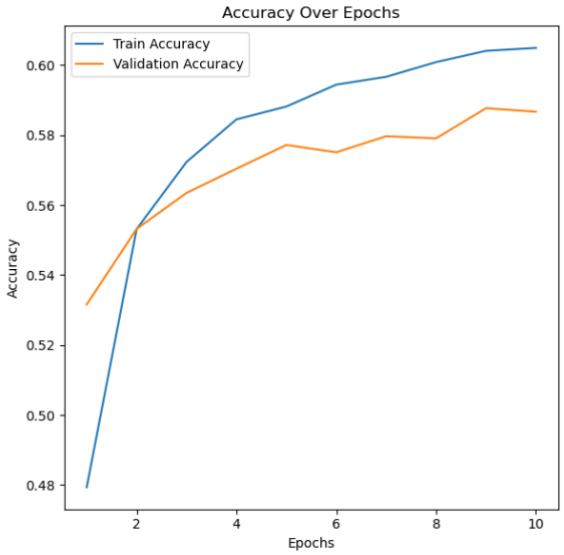
After fine-tuning, the model achieved a test accuracy of 73.72%. Precision, recall, and F1 score metrics were consistent with the test accuracy, indicating balanced and reliable performance across different classes.

Performance Plots

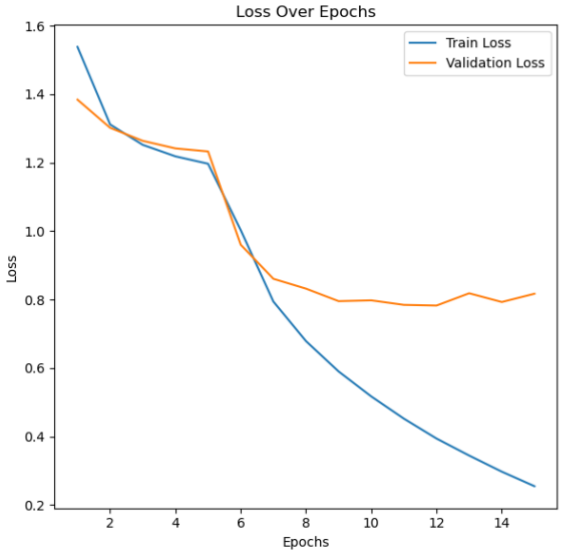
The following plots illustrate the loss and accuracy trends over the epochs for both the initial and fine-tuning phases:

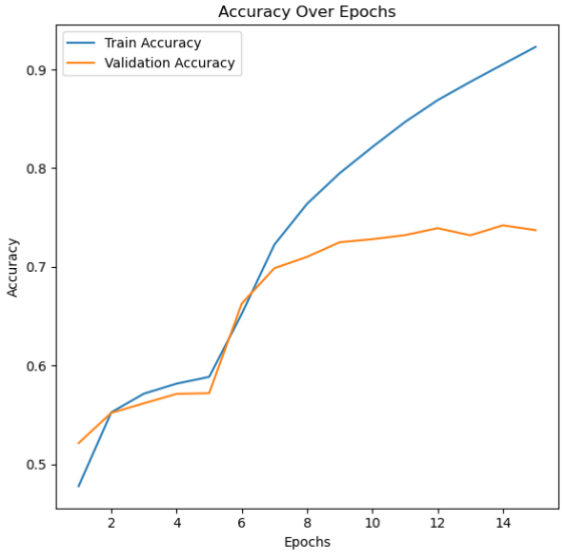
Initial phase:





Fine-tuning phase





20. Discussion

Insights and Observations

1. **Effective Feature Extraction**: The Custom Model, with its pre-trained weights, served as an effective feature extractor in the initial phase.
2. **Significance of Fine-Tuning**: Unfreezing and retraining the last 4 layers of the Custom Model significantly enhanced its performance for the CIFAR-10 dataset.
3. **Balanced Metrics**: The final evaluation metrics indicated that the model performed consistently across different classes, making it suitable for diverse image classification tasks.

Model Usage

This fine-tuned model is well-suited for tasks similar to CIFAR-10. The approach of starting with a frozen model and gradually fine-tuning it is beneficial for adapting pre-trained models to new, specific tasks with limited data.

Limitations

- **Computational Demands**: Fine-tuning increases the demand for computational resources and time.
- **Risk of Overfitting**: Fine-tuning can lead to overfitting, especially when dealing with small datasets.

Future Improvements

1. **Extensive Fine-Tuning**: Further unfreezing more layers or even the entire model could yield better performance, although it increases the risk of overfitting.
2. **Data Augmentation**: Implementing data augmentation techniques could help the model generalize better to unseen data.
3. **Regularization**: Adding dropout or L2 regularization in the custom layers can help mitigate overfitting during fine-tuning.

21. Conclusion

The transfer learning and fine-tuning approach with Custom Model on the CIFAR-10 dataset demonstrated the effectiveness of using pre-trained models for new tasks. By initially training with frozen layers and then selectively fine-tuning deeper layers, we achieved significant improvements in model performance. This method provides a robust framework for applying transfer learning to various image classification challenges.

IronHackers Students

Fernando Nuno Vieira
Kour Nashto