POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY 1

PYTHON PROGRAMMING

ASSIGNMENT 2

REPORT OF IMAGE PROCESSING WITH CIFAR-10 DATASET

Instructor: Kim Ngoc Bach
Student: Nguyen Manh Hung
Student ID: B23DCCE040
Class ID: D23CQCE04-B

Hanoi

Table of contents

Chapter 1

Introduction

1.1 Abstract

This report focuses on image classification using the CIFAR-10 dataset. The task is to implement and compare two types of neural networks: a basic 3-layer Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN) (in my assignment, both models are created with optional number of layers). The project involves the complete machine learning workflow, including training, validation, and testing of both models. Performance evaluation will be conducted by plotting learning curves and confusion matrices. A key component of the assignment is a comparative analysis of the results obtained from the MLP and CNN architectures.

1.2 Goal

The goal of this assignment is to develop and compare the performance of a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN) for image classification on the CIFAR-10 dataset using PyTorch.

This involves:

- Building a basic 3-layer MLP (I built more than 3 layers for better efficiency).
- Building a CNN with multiple convolution layers (I used MobileNet-V2 instead)
- Training, validating, and testing both neural networks.
- Evaluating their performance using learning curves and confusion matrices.
- Analyzing and discussing the results to compare the effectiveness of the two architectures for this task.

Chapter 2

Background Knowledge

2.1 Multi-layer Perceptron model

Multi-Layer Perceptron (MLP) is an artificial neural network widely used for solving classification and regression tasks. MLP consists of fully connected dense layers that transform input data from one dimension to another. It is called "multi-layer" because it contains an input layer, one or more hidden layers, and an output layer. The purpose of an MLP is to model complex relationships between inputs and outputs, making it a powerful tool for various machine learning tasks.

2.1.1 Key Components of Multi-Layer Perceptron (MLP)

- **Input Layer**: Each neuron (or node) in this layer corresponds to an input feature. For instance, if you have three input features, the input layer will have three neurons.
- **Hidden Layers**: An MLP can have any number of hidden layers, with each layer containing any number of nodes. These layers process the information received from the input layer.
- **Output Layer**: The output layer generates the final prediction or result. If there are multiple outputs, the output layer will have a corresponding number of neurons.

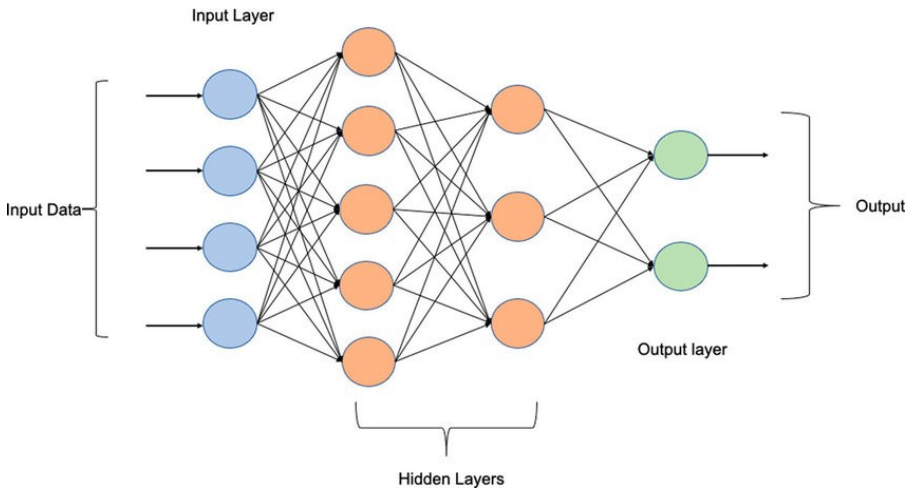


Figure 2.1: MLP neural network with 2 hidden layers

2.1.2 Working of Multi-Layer Perceptron

2.1.3 Step 1: Forward Propagation

In **forward propagation**, the data flows from the input layer to the output layer, passing through any hidden layers. Each neuron in the hidden layers processes the input as follows:

Weighted Sum

The neuron computes the weighted sum of the inputs:

$$z = \sum_i w_i x_i + b$$

Where:

- x_i is the input feature.
- w_i is the corresponding weight.
- b is the bias term.

Activation Function

The weighted sum z is passed through an activation function to introduce non-linearity. Common activation functions include:

- **Sigmoid**: $\sigma(z) = \frac{1}{1+e^{-z}}$
- **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$ (or using `amsmath`: $f(z) = \max(0, z)$)
- **Tanh (Hyperbolic Tangent)**: $\tanh(z) = \frac{2}{1+e^{-2z}} - 1$ (or using `amsmath`: $\tanh(z) = \frac{2}{1+e^{-2z}} - 1$)

Note

In my assignment, I replace ReLU activation function with GELU (Gaussian Error Linear Unit), allowing (albeit down-weighted) negative values to maintain better gradient flow, even for negative inputs. This can help the model learn more complex representations and mitigate the dying ReLU problem

In greater detail, the Gaussian Error Linear Unit, or GELU, is an activation function. The GELU activation function is $x\Phi(x)$, where $\Phi(x)$ is the standard Gaussian cumulative distribution function. The GELU nonlinearity weights inputs by their percentile, rather than gates inputs by their sign as in ReLUs ($x\mathbf{1}_{x>0}$). Consequently the GELU can be thought of as a smoother ReLU.

$$\text{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right],$$

if $X \sim N(0, 1)$.

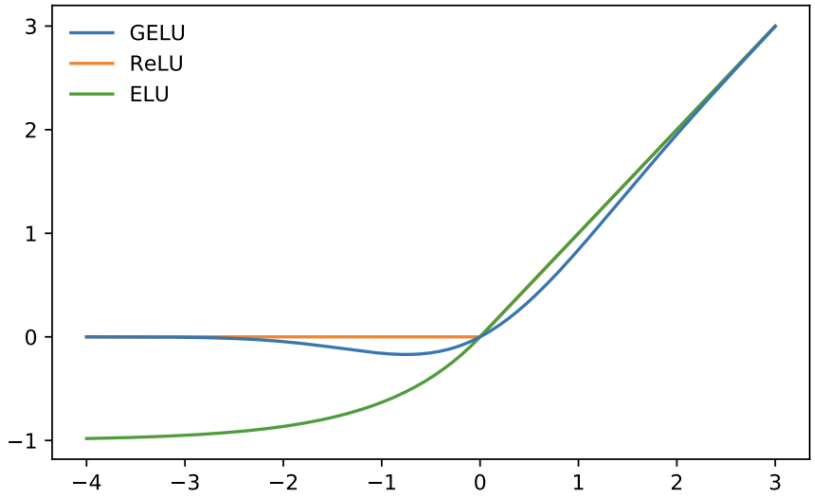


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Figure 2.2: Visualization of GELU

2.1.4 Step 2: Loss Function

Once the network generates an output, the next step is to calculate the loss using a **loss function**. In supervised learning, this compares the predicted output to the actual label.

For a classification problem, the commonly used **binary cross-entropy** loss function is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Where:

- y_i is the actual label.
- $\hat{y}_i$ is the predicted label.
- N is the number of samples.

For regression problems, the **mean squared error (MSE)** is often used:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

2.1.5 Step 3: Backpropagation

The goal of training an MLP is to minimize the loss function by adjusting the network's weights and biases. This is achieved through **backpropagation**:

Gradient Calculation: The gradients of the loss function with respect to each weight and bias are calculated using the chain rule of calculus.

Error Propagation: The error is propagated back through the network, layer by layer.

Gradient Descent: The network updates the weights and biases by moving in the opposite direction of the gradient to reduce the loss:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

Where:

- w is the weight.
- η is the learning rate.
- $\frac{\partial L}{\partial w}$ is the gradient of the loss function with respect to the weight.

2.1.6 Step 4: Optimization

MLPs rely on optimization algorithms to iteratively refine the weights and biases during training. Popular optimization methods include:

Stochastic Gradient Descent (SGD): Updates the weights based on a single sample or a small batch of data:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

Adam Optimizer: An extension of SGD that incorporates momentum and adaptive learning rates for more efficient training:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

Here, g_t represents the gradient at time t, and β_1, β_2 are decay rates.

2.2 Mobilenet-V2 model

MobileNetV2 is a convolutional neural network architecture optimized for mobile and embedded vision applications. It improves upon the original MobileNet by introducing inverted residual blocks and linear bottlenecks, resulting in higher accuracy and speed while maintaining low computational costs. MobileNetV2 is widely used for tasks like image classification, object detection, and semantic segmentation on mobile and edge devices.

Architecture of MobileNet V2

The MobileNet V2 architecture is designed to provide high performance while maintaining efficiency for mobile and embedded applications. Below, we break down the architecture in detail, using the schematic of the MobileNet V2 structure as a reference.

2.2.1 Initial Layers

- **Input Layer:** The model takes an RGB image of fixed size (224×224 pixels) as input.
- **First Convolutional Layer:** This layer applies a standard convolution with a stride of 2 to downsample the input image. This operation increases the number of channels to 32.

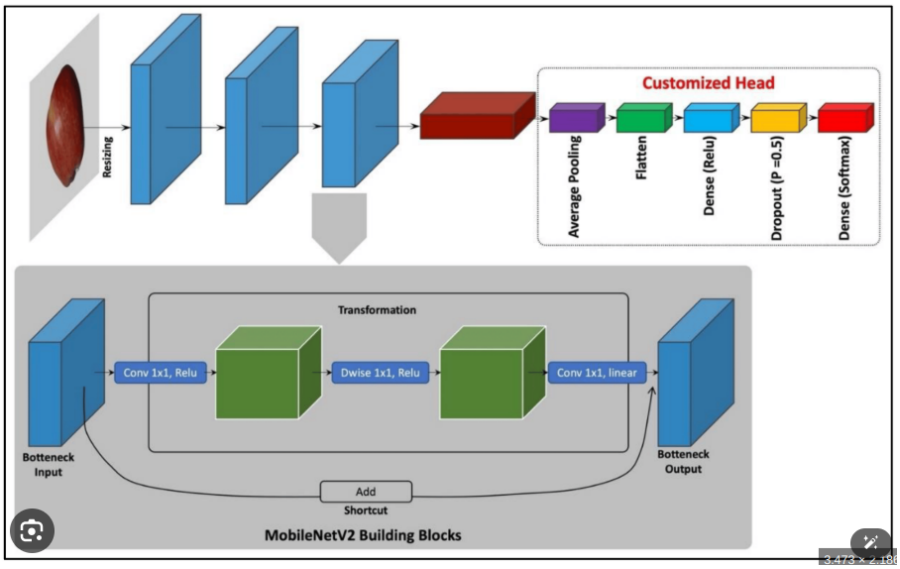


Figure 2.3: Blocks of MobileNet-V2

2.2.2 Inverted Residual Blocks

The core component of MobileNet V2 is the inverted residual block, which consists of three main layers:

- **Expansion Layer:** A 1×1 convolution that increases the number of channels (also known as the expansion factor). This layer is followed by the ReLU6 activation function, which introduces non-linearity.
- **Depthwise Convolution:** A depthwise convolution layer that performs spatial convolution independently over each channel. This layer is also followed by ReLU6.
- **Projection Layer:** A 1×1 convolution that projects the expanded channels back to a lower dimension. This layer does not use an activation function, hence it is linear.

Each inverted residual block has a shortcut connection that skips over the depthwise convolution and connects directly from the input to the output, allowing for better gradient flow during training. This connection only exists when the input and output dimensions match.

2.3 Detailed Structure of Inverted Residual Blocks

First Block: The initial block after the first convolution has a stride of 1 and does not perform downsampling. It has an expansion factor of 1 and 16 output channels.

Subsequent Blocks: The following blocks have varying strides and expansion factors:

- The first set of blocks (with a stride of 2) reduces the spatial dimensions of the input.
- Each subsequent block applies the expansion, depthwise convolution, and projection layers.
- The expansion factor is typically set to 6 for most blocks.

2.3.1 Specific Block Details

The architecture can be summarized in a table format, where each line describes a sequence of identical (modulo stride) layers, for example:

Table 2.1: MobileNet V2 Block Specifications

Input Size	Operator	Exp. Factor (t)	Out Channels (c)	Repeats (n)	Stride (s)
$224 \times 224 \times 3$	Conv2D	-	32	1	2
$112 \times 112 \times 32$	Bottleneck	1	16	1	1
$112 \times 112 \times 16$	Bottleneck	6	24	2	2
$56 \times 56 \times 24$	Bottleneck	6	32	3	2
$28 \times 28 \times 32$	Bottleneck	6	64	4	2
$14 \times 14 \times 64$	Bottleneck	6	96	3	1
$14 \times 14 \times 96$	Bottleneck	6	160	3	2
$7 \times 7 \times 160$	Bottleneck	6	320	1	1
$7 \times 7 \times 320$	Conv2D	-	1280	1	1
$7 \times 7 \times 1280$	AvgPool 7×7	-	-	1	-
$1 \times 1 \times 1280$	Conv2D	-	k (classes)	1	-

2.3.2 Final Layers

Conv2D Layer: After the series of inverted residual blocks, a final 1×1 convolution layer increases the channel dimensions to 1280.

Average Pooling Layer: A global average pooling layer reduces the spatial dimensions to 1×1, producing a feature vector.

Fully Connected Layer: The final layer is a fully connected layer that outputs the class scores for classification tasks.

Chapter 3

Data preprocessing

For the model **Multi-layer Perceptron** (implemented in `main_MLP.ipynb`), the data preparation process involved the following key steps:

1. **Loading and Initial Data Formatting:**
    - CIFAR-10 data was loaded from the original batch files.
    - Raw image data, initially a flat vector per image, was reshaped into 3-dimensional NumPy arrays with HWC (Height-Width-Channel) format (e.g., $N \times 32 \times 32 \times 3$), using a `uint8` data type.
    - This process yielded the initial datasets: `x_train`, `y_train`, `x_test`, and `y_test`.
2. **PyTorch Data Preparation:**
    - A custom PyTorch `Dataset` class was implemented to handle the CIFAR-10 data.
    - **Transformations for Training Data:** These included:
        - Conversion of images (NumPy HWC, `uint8`) to PyTorch tensors (CHW – Channel-Height-Width – format, scaled to the $[0.0, 1.0]$ range) using `transforms.ToTensor()`.
        - Normalization of image tensors using CIFAR-10 specific mean and standard deviation values via `transforms.Normalize()`.
        - Application of various data augmentation techniques: random crops, random horizontal flips, random rotations, random color jitter, and random erasing.
    - **Transformations for Validation and Test Data:** These included only the conversion to PyTorch tensors and normalization, excluding random data augmentations to ensure consistent evaluation.
    - Labels were converted to PyTorch LongTensors, suitable for the `CrossEntropyLoss` function.
3. **Data Splitting and DataLoader Creation:**
    - The initial training set (`x_train`, `y_train`) was further divided into a new training set and a validation set. This facilitates monitoring model performance during training and aids in hyperparameter tuning.
    - Finally, PyTorch `DataLoader` instances (`train_loader`, `val_loader`, and `test_loader`) were created. These DataLoaders provide data in batches, automatically shuffle the training data, and can utilize multiple worker processes for efficient data loading for model training, validation, and testing phases.

After these steps, the data was prepared and appropriately formatted for input into the MLP model training process.

Figure 3.1: Result after preprocessing data for MLP

For the model **MobileNetV2**, the data preparation process (implemented in `main_MBN.ipynb`) is nearly the same as that of the MLP, but there are some key differences:

- **Flattening images**: While the MLP model expects a flat (1D) feature vector for each data sample, the MobileNetV2 model takes input as multi-dimensional image data (e.g., `[batch_size, channels, height, width]`).
- **Resizing images**: Images are resized to 224×224 to match the architecture of MobileNetV2 (which is typically trained on ImageNet at this size). This ensures that the convolutional and pooling layers operate as intended.

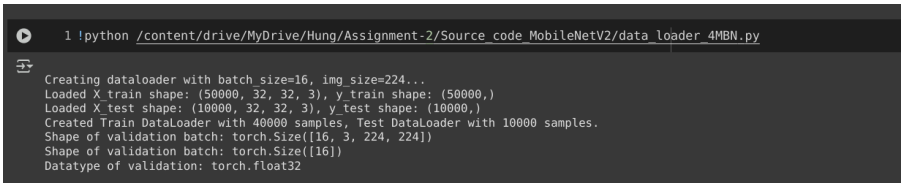


Figure 3.2: Result after preprocessing data for MobileNet-V2

Chapter 4

Defining the models

4.1 Multi-layer Perceptron

The code for this module is implemented in `main_MLP.ipynb`

Input Processing

The model is designed to accept input images with dimensions $3 \times 32 \times 32$ (representing 3 color channels and 32×32 pixels). In the initial step of the `forward` pass, these 3D image tensors are flattened into 1D vectors. Each vector comprises $3 \times 32 \times 32 = 3072$ features.

Hidden Layers

The network incorporates four hidden layers. Each hidden layer block sequentially applies the following operations:

1. **Linear Transformation**
2. **Batch Normalization**
3. **GELU Activation**
4. **Dropout**

The dimensions (number of neurons) of the hidden layers follow a narrowing funnel structure:

- **Hidden Layer 1**: Input from 3072 features, output to 512 neurons.
- **Hidden Layer 2**: Input from 512 features, output to 256 neurons.
- **Hidden Layer 3**: Input from 256 features, output to 128 neurons.
- **Hidden Layer 4**: Input from 128 features, output to 64 neurons.

Output Layer

The final layer is a single linear transformation (`nn.Linear`, referred to as `self.fc_output` in the implementation).

- It takes the 64 features output by the fourth hidden layer (after its dropout stage).
- It maps these features to `num_classes = 10` output neurons.

- Each output neuron produces a raw score (logit) corresponding to one of the target classes. No activation function is applied directly to these logits within the model, as this is typically integrated into the loss function (e.g., `CrossEntropyLoss`) during training.

Implementation Details

The model is implemented as a Python class, `MLP`, inheriting from `torch.nn.Module`. The constituent layers are defined and initialized within the `__init__` method. The sequence of operations and data flow through the network is specified in the `forward` method.

4.2 MobileNet-V2

The code for this module is implemented in `main_MBN.ipynb`

4.2.1 Core Architectural Features

The architecture is founded on these components:

- **Inverted Residual Blocks:** These are the fundamental building units. Each block typically involves:
  - An initial 1x1 convolutional layer: to expand channels
  - A 3x3 depthwise convolution: for spatial filtering
  - A 1x1 pointwise convolutional layer: to project channels down
  - Residual connections where applicable
- **ReLU6 Activation**
- **Batch Normalization (`nn.BatchNorm2d`):**

4.2.2 Key Layers & Overall Flow

The model processes input data through the following sequence of operations:

1. **Initial Convolutional Layer:** A `ConvBNReLU` block (3x3 Convolution, BatchNorm, ReLU6). It accepts 3 input channels (standard RGB). The number of output channels is `_make_divisible`$(32 \times$ `width_mult`$, 8)$. This layer uses a fixed `stride=2`.
2. **Sequence of Inverted Residual Blocks:** A stack of 'InvertedResidual' blocks, the configurations of which (expansion ratio `t`, base output channels `c`, number of repetitions `n`, and initial stride `s`) are defined by an internal `inverted_residual_setting` table. The output channels for each stage are scaled by `width_mult`. Blocks with `s=2` perform spatial downsampling.
3. **Final 1x1 Convolutional Layer:** Another `ConvBNReLU` block using a 1x1 kernel. This layer expands the feature channels to `_make_divisible`$(1280 \times$ `width_mult`$, 8)$.
4. **Global Average Pooling:** An `nn.AdaptiveAvgPool2d((1, 1))` layer reduces each feature map to a single value, producing a feature vector.
5. **Classifier Head:** An `nn.Sequential` block consisting of an `nn.Dropout(0.2)` layer (hardcoded rate) followed by an `nn.Linear` layer mapping the feature vector to `num_classes` output scores.

4.2.3 Key Customizable Parameters

The model's instantiation can be configured using the following parameters:

- `num_classes`: Specifies the number of output classes for the classifier (default: 10).
- `width_mult`: A floating-point multiplier that scales the number of channels across all layers (default: 1.0).
- `classifier_dropout_rate`: A parameter intended for the classifier's dropout rate (default: 0.1 in the factory function).
- `input_size`: A parameter intended to potentially adjust model aspects based on input image dimensions.
- `norm_layer`: Allows specification of the normalization layer (default: `nn.BatchNorm2d`).

Chapter 5

Training and Evaluation Details

Note:

- The results for MLP are taken from `/Assignment-2/Experiments/MLP_run/Exp20`
- The results for MobileNet-V2 are taken from `/Assignment-2/Experiments/MobileNetV2_cifar_run/Try_4`

5.1 Learning Curve

Regarding MLP

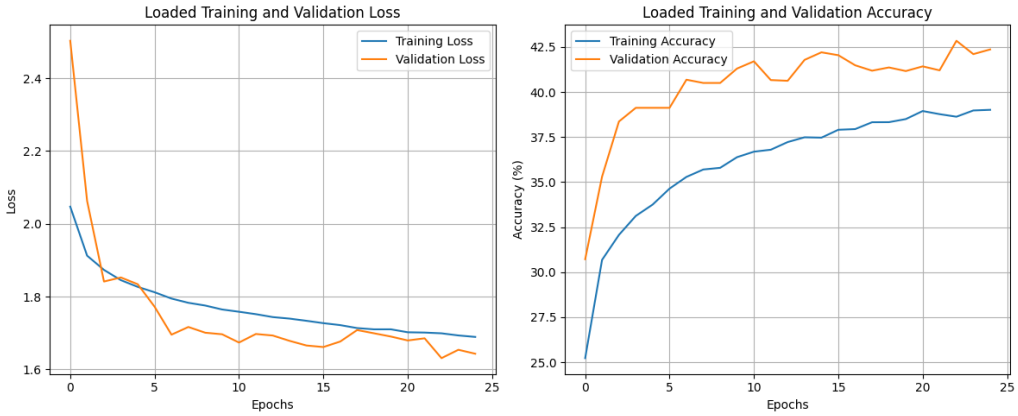


Figure 5.1: Learning Curve of Multilayer Perceptron on the CIFAR-10 Dataset

Key observations

- Underfitting is present, which is shown by accuracy and lost performances.
  - **Accuracy**: The validation accuracy peaks and even slightly degrades after around epoch 15, hovering between 41-42.5%, while the training accuracy continues to climb, approaching 40%. This divergence indicates that the regularization effect is too aggressive, which includes dropout, batch normalization in each hidden layer, and AdamW's weight decay. Therefore, the model is not able to learn the training data in a detailed manner, showing a slight sign of underfitting but achieving better generalization on new, unseen data.

  - **Loss**: While the training loss gradually decreases from epoch 1, the validation loss significantly drops to well below the training loss after epoch 5 and then fluctuates at a moderate level, occasionally approaching it again at certain epochs. This also indicates good generalization of the model on unseen data, corresponding to accuracy performance.

- Low performance: CIFAR-10 is a dataset of 32x32 pixel images. MLPs, by design, flatten the image input (x.view(x.size(0), -1)), which destroys the spatial relationships between pixels (e.g., what pixels are next to each other). Therefore, the validation accuracy is just up to around 42%.

Regarding MobileNet-V2 and Comparison to MLP

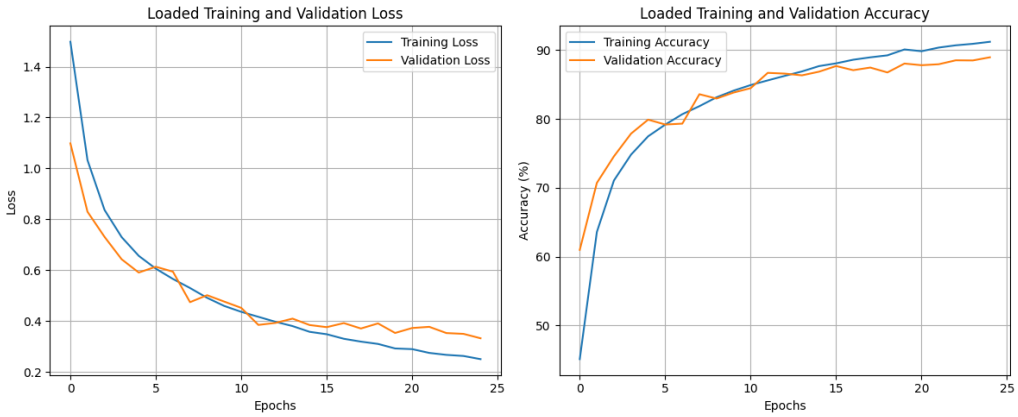


Figure 5.2: Learning Curve of MobileNet-V2 on the CIFAR-10 Dataset

Key observations

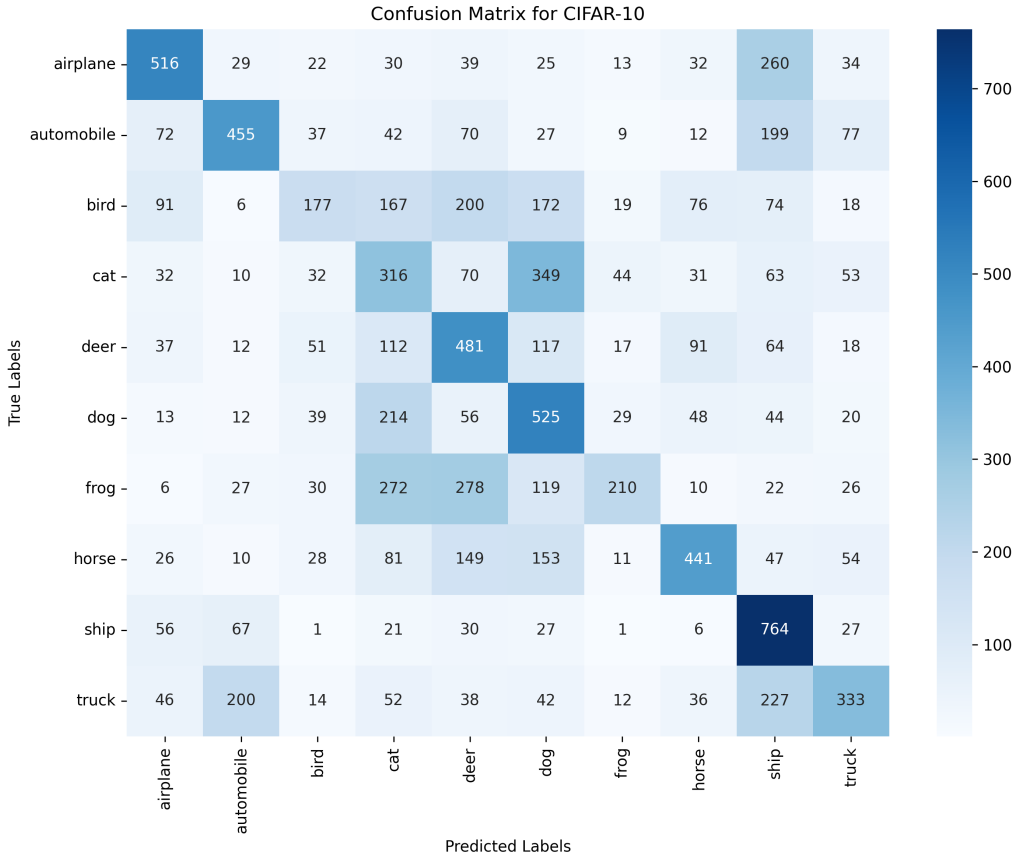
- **Good fit**: The training and validation curves for both loss and accuracy track each other very closely. In comparison to the great divergence between training and validation loss, and significant fluctuation of validation accuracy in MLP performance, the gap between the training and validation lines in MobileNet-V2 is minimal and stable, showing no sign of underfitting, in details:
  - **Residual Connection (Shortcut)**: helps to combat vanishing gradient and promotes stable learning
  - **Fewer parameters**: Depth-wise convolution forces the model to learn the most important and generalizable patterns, preventing it from noise memorization.
- **Strong performance**: The model achieves a high validation accuracy, reaching just under 90% in 25 epochs, comparing to just nearly approaching 40% in previous MLP. To be more specific:
  - **Deeper is better**: The extreme efficiency of depthwise convolutions means I can stack many more of these blocks to create a much deeper network to learn more complex and abstract feature hierarchies while keeping computational cost low.
  - **Richer feature**: By expanding the input to a high-dimensional space, the network has a larger space to learn complex relationships between channels. Even though this is temporary, it allows the model to capture more intricate patterns before squeezing the result back down.

5.2 Confusion Matrix

Regarding MLP

Analysis

The provided confusion matrix for the CIFAR-10 classification task indicates that the model has achieved a moderate level of accuracy. The diagonal entries representing correct predictions are consistently the largest in their respective rows. However, the magnitude of several off-diagonal values reveals critical confusion that degrades overall performance.



High-performing classes

The model shows a high degree of competence in identifying ships, correctly classifying 764 instances with very few misclassifications. This suggests the model has effectively learned the unique features of this class. The airplane class also performs well with 516 correct predictions.

Key areas of confusion

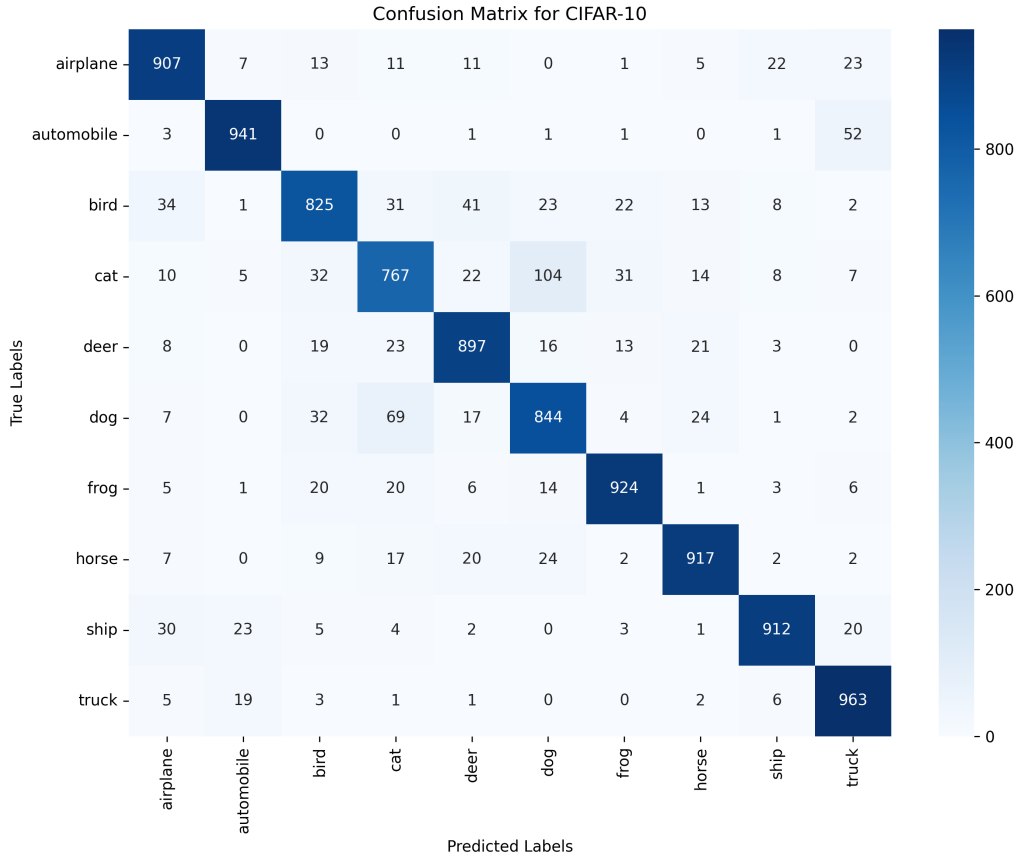
- **Inter-Animal Confusion**: The most significant error is the confusion between cats and dogs. The model misclassifies 349 'cat' images as 'dog', and conversely, 214 'dog' images as 'cat'. This classic problem highlights the model's difficulty with fine-grained distinctions between similar animal shapes and textures. The 'bird' class is also very poorly classified, often being mistaken for airplanes (91), cats (167), and dogs (172).

- **Inter-Vehicle Confusion**: A similar issue exists between ground vehicles. The model frequently mislabels 'trucks' as 'automobiles' (227 instances) and shows some confusion in the other direction as well (72 instances).
- **Cross-Category Confusion**: An interesting and significant error is the misclassification of 260 'ships' as 'airplanes'. This could be due to the model over-relying on the presence of a blue background (water or sky) as a primary feature for both classes.

Regarding MobileNet-V2

Analysis

The provided confusion matrix for the CIFAR-10 classification task demonstrates a highly effective and well-performing model. The diagonal entries, which represent correct classifications, are notably high across all ten classes, indicating that the model has learned to distinguish between the categories with a strong degree of accuracy. Off-diagonal values representing misclassifications are generally low, further underscoring the model's proficiency.



Overall Performance and Strengths:

- **Trucks**: 963 correct classifications, with minimal confusion.
- **Automobiles**: 941 correct classifications, with the primary minor confusion being with 'truck' (52 instances).
- **Frogs**: 924 correct classifications.

- **Horses**: 917 correct classifications.
- **Ships**: 912 correct classifications.
- **Airplanes**: 907 correct classifications.
- **Deers**: 897 correct classifications.
- **Dogs**: 844 correct classifications.

Key areas of confusion

- **Cat vs. Dog**: This remains the most noticeable area of confusion, though at a much-reduced scale compared to less optimized models. 104 instances of 'cat' were misclassified as 'dog', and 69 instances of 'dog' were misclassified as 'cat'. This highlights that differentiating these two animals remains a subtle challenge.
- **Bird Misclassifications**: The 'bird' class, while still achieving a high 825 correct predictions, shows some scattered confusion with 'airplane' (34 instances), 'deer' (41 instances), and 'cat' (31 instances). This suggests that features related to sky or certain animal shapes might occasionally overlap.
- **Vehicle Sub-Types**: As mentioned, 'automobile' is sometimes confused with 'truck' (52 instances), which is understandable given their shared context and features.

Comparison

- Model MLP shows moderate performance. While the diagonal values (correct classifications) are generally the highest in their respective rows, there are very significant off-diagonal values, indicating widespread confusion between many classes. The overall accuracy suggested by MLP would be considerably lower than MobileNet-V2.
- Model mobileNet-V2 demonstrates a highly effective and accurate model. The diagonal values are substantially higher across all classes compared to MLP, and the off-diagonal values (misclassifications) are drastically reduced. This indicates a model that has learned to differentiate between the classes with much greater precision. Visually, the heatmap for MobileNet-V2 shows a much darker, more defined diagonal and significantly lighter off-diagonal cells, which is characteristic of a high-performing classifier.

Conclusion

The comparison unequivocally demonstrates that model MobileNet-V2 is substantially more accurate and reliable than model MLP. In terms of learning curves comparison, MobileNet-V2 is much more stable and has better fit between training and validation lines than MLP. Also, it has learned to distinguish the CIFAR-10 classes with high precision, drastically reducing both the major, well-known confusions (like cat/dog) and the widespread, less specific misclassifications that plagued MLP. This suggests significant enhancements in the model architecture, training process for MobileNet-V2 compared to MLP.