POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY I

o0o

ASSIGNMENT 2 REPORT

PYTHON PROGRAMMING LANGUAGE

Instructor:	**Kim Ngoc Bach**
Student:	**Phung Thu Huong**
Student ID:	**B23DCVT201**
Class:	**D23CQCEO6-B**
Academic Year:	**2023 - 2028**
Training System:	**Full-time University**

Hanoi, 2025

POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY I

—o0o—

ASSIGNMENT 2 REPORT
PYTHON PROGRAMMING LANGUAGE

Instructor: Kim Ngoc Bach
Student: Phung Thu Huong
Student ID: B23DCVT201
Class: D23CQCEO6-B
Academic Year: 2023 - 2028
Training System: Full-time University

Hanoi, 2025

SUPERVISOR'S COMMENTS

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Score: **(In words:**)

Hanoi, day month year 20...

Supervisor

Contents

List of Figures

Introduction

In the field of Computer Vision, image classification is one of the most fundamental and crucial tasks. The objective is to assign a specific label to an image based on its content. For instance, distinguishing between cats, dogs, airplanes, cars, etc.

While humans can easily identify objects in images, teaching computers to do so poses a significant challenge. Traditional methods often require manual feature extraction, which is time-consuming and ineffective given the complexity of image data.

The problem statement: How can we build a machine learning model capable of automatically learning features from images and classifying them accurately?

Reason for processing: To address this problem, Convolutional Neural Networks (CNNs) have emerged as a powerful solution. CNNs can automatically learn hierarchical features from image data through convolutional layers, making them superior in computer vision tasks compared to traditional neural networks.

This assignment focuses on:

- Building a CNN model with 3 convolutional layers.
- Performing the training, validation, and testing processes.
- Visualizing the learning process through Learning Curves.
- Evaluating detailed performance using a Confusion Matrix.

The entire process will be implemented using the PyTorch library, a powerful and flexible deep learning framework.

Chapter 1

Image Processing Logic (Overview)

The general logic for image processing for classification using CNNs in this assignment includes the following steps:

1.1 Data Preprocessing & Augmentation

The first stage focuses on preparing the input data from the CIFAR-10 dataset. Images, with dimensions of 32×32 pixels and 3 color channels, undergo normalization to ensure stability during training. Data augmentation techniques, including random crop, horizontal flip, and random erasing of an image region, are applied. These augmentation methods help the model learn invariant features and mitigate overfitting.

1.2 CNN Architecture with 3 Convolutional Layers

A CNN model comprising 3 convolutional layers has been constructed. After preprocessing, image data is passed through a sequence of layered convolutional blocks. Each convolutional layer is combined with a **ReLU** activation function and a **Pooling** layer to form a complete processing unit. This architecture is designed to extract features sequentially from low-level attributes like edges and corners, progressing to more complex features such as object parts and overall shapes. To enhance stability and optimize learning speed, **Batch Normalization** layers are integrated into the network architecture.

1.3 Classification and Output Layer

In the classification stage, features extracted from the 3 convolutional layers are transformed into a one-dimensional vector through a **flattening** process. This feature vector is then processed through **Fully Connected layers** to perform the final classification. To enhance generalization capabilities, a **Dropout** layer is integrated to control and mitigate the risk of overfitting. The final output layer returns a probabilistic prediction distribution for the 10 object classes in the CIFAR-10 dataset.

Chapter 2

Implementation Logic Sequence

2.1 Libraries Used

To build and train an efficient CNN for the image classification task, the following libraries were selected based on their ability to meet the requirements:

- **os**: For interacting with the operating system (creating directories, managing file paths).
- **torch, torchvision, torch.nn, torch.nn.functional, torch.optim, torch.optim.lr_sched** The core PyTorch foundation, providing tensor structures with automatic GPU detection and utilization, network layers (`Conv2d`, `Linear`, `BatchNorm2d`), optimization algorithms (`AdamW`), and dynamic Learning Rate adjustment mechanisms (`ReduceLROnPlateau`).
- **torchvision.transforms**: Image preprocessing and augmentation tools (`Normalize`, `RandomCrop`, `RandomHorizontalFlip`, `RandomErasing`).
- **matplotlib.pyplot, numpy, seaborn**: For visualizing learning curves and confusion matrices, and numerical data processing.
- **sklearn.metrics.confusion_matrix**: For calculating the confusion matrix.
- **time, datetime, sys**: Used for logging the training process and monitoring time.
- **torch.utils.data.random_split**: For splitting datasets into random subsets.

2.2 Hyperparameters Configuration and Initial Setup

The deep learning model training process depends on various hyperparameters and environmental settings. The main configurations are defined centrally to ensure flexibility, reproducibility of results, and optimal utilization of hardware resources:

- **Hardware Configuration**:
  - **DEVICE**: Automatically detects GPU (`cuda:0`) or CPU.
  - **PIN_MEMORY**: Automatically enabled when a GPU is present to optimize data transfer.
- **Training Hyperparameters**:

  - **BATCH_SIZE** = 128: Batch size balancing efficiency and memory.
  - **LR** = 0.001: Initial learning rate for `AdamW`.
  - **EPOCHS** = 300: Maximum number of epochs allowed.

- **Adjustment and Early Stopping Mechanism**:
  - **PATIENCE** = 5: Early stopping after 5 epochs without improvement in validation loss.

- **Regularization Techniques**:
  - **DROPOUT_RATE** = 0.5: High dropout rate to prevent overfitting.
  - **LABEL_SMOOTHING** = 0.1: Label smoothing to improve generalization.
  - **WEIGHT_DECAY** = 5e-4: Increased L2 regularization to combat overfitting.

- **Reproducibility**:
  - `torch.manual_seed(42)`, `np.random.seed(42)`: Fixes the seed to ensure consistent results across runs.

2.3 Logging and Directory Setup

To manage and monitor the training process, the following functions are implemented:

- `setup_logging()`: Creates the `logs` directory and configures the logging system to store detailed training information in a log file and simultaneously display it on the console.
- `create_directories()`: Creates `results`, `models`, and `plots` directories to organize and store outputs such as trained models and result plots.

2.4 Sample Image Display Function (imshow)

Purpose: To visualize sample images from the dataset after preprocessing. Implementation: This function takes an image tensor and a filename for saving. It converts the image tensor to NumPy format, then rescales pixel values to the $[0, 1]$ range (as images were normalized to $[-1, 1]$) and displays them. The results are saved to the `plots` directory.

2.5 CNN Model Definition (CNN Class)

The model is designed based on the principle of balancing representational capacity and practicality, deep enough to learn complex features but not overly complex to avoid overfitting and resource consumption.

- **Overall Structure**: The CNN class inherits from `nn.Module`, comprising two main parts: three convolutional blocks for feature extraction and two fully connected layers for classification. This architecture gradually increases the number of channels from 3 to 32, 64, then 128 channels, while progressively reducing spatial dimensions from 32×32 to 16×16, 8×8, and finally 4×4.

- **Convolutional Block**: Each convolutional block includes:
  - `nn.Conv2d`: Performs convolution to extract features. A 3×3 kernel size and `padding=1` are used to preserve spatial dimensions.
  - `nn.BatchNorm2d`: Normalizes the output of the convolutional layer, aiding in stabilization and accelerating training.
  - `nn.MaxPool2d`: Reduces the spatial dimensions of the feature maps, helping to decrease the number of parameters and increase invariance to translation.
- **Classification Layer**:
  - `nn.Linear(128 * 4 * 4, 512)`: Converts the output of the convolutional layers into a flattened vector, then maps it to 512 features.
  - `nn.BatchNorm1d(512)`: Normalizes the output of the first Fully Connected layer.
  - `nn.Dropout(dropout_rate)`: Applied after the first Fully Connected layer to randomly "turn off" neurons, preventing overfitting.
  - `nn.Linear(512, num_classes)`: The final output layer, mapping to the number of classes to be classified (10 classes for CIFAR-10).
- **Forward Pass Flow**: Data passes through three convolutional blocks in a fixed pattern: `Convolution` $\rightarrow$ `BatchNorm` $\rightarrow$ `ReLU` $\rightarrow$ `MaxPool`. Subsequently, the data is flattened (`x.view(-1, ...)`) and passed through two fully connected layers with dropout applied in between. The `F.relu` activation function is used to introduce non-linearity.

2.6 Data Loading and Preprocessing (get_data_loaders)

This function is responsible for loading the CIFAR-10 dataset and preparing it for training, validation, and testing.

- **Data Transformations (transforms.Compose)**:
  - `transform_train`: Applies data augmentation transformations for the training set such as `RandomCrop`, `RandomHorizontalFlip`, `RandomErasing` to increase diversity and prevent overfitting. Finally, `ToTensor()` and `Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))` are applied to normalize pixels to the $[-1, 1]$ range.
  - `transform_test`: Only applies `ToTensor()` and `Normalize()` to the validation and test sets to ensure objective evaluation.
- **Dataset Loading**: Uses `torchvision.datasets.CIFAR10` to load the data.
- **Training/Validation Split**: `full_train` is split into `train_set` (80%) and `val_set` (20%) using `random_split`.
- **DataLoader**: Creates `DataLoader` for `train_set`, `val_set`, and `test_set` with `batch_size`, `shuffle` (only for the train set), `num_workers` (for parallel data loading), and `pin_memory` to optimize data transfer to the GPU.

2.7 Model Training Function (train_model)

This function executes the main training loop, including the forward/backward pass, weight updates, and performance tracking on both the training and validation sets.

- **Epoch Loop**: Iterates through the predefined number of epochs.
- **Training Phase** (`model.train()`):
  - Sets the model to training mode (enables `Dropout`, `BatchNorm` updates statistics).
  - Iterates through each batch of data from `train_loader`.
  - `optimizer.zero_grad()`: Clears old gradients.
  - `model(inputs)`: Performs the forward pass.
  - `criterion(outputs, targets)`: Calculates the loss.
  - `loss.backward()`: Backpropagates to compute gradients.
  - `optimizer.step()`: Updates weights.
  - Aggregates `train_loss` and `train_acc`.
- **Validation Phase** (`model.eval()`):
  - Switches the model to evaluation mode (disables `Dropout`, `BatchNorm` uses fixed statistics).
  - `with torch.no_grad()`: Disables gradient computation to save memory and accelerate.
  - Iterates through each batch of data from `val_loader`, calculating `val_loss` and `val_acc`.
- **Learning Rate Adjustment**: `scheduler.step(val_loss)` adjusts the learning rate based on `val_loss`.
- **History Logging**: Stores `train_loss`, `val_loss`, `train_acc`, `val_acc` into `history`.
- **Early Stopping**: Compares the current `val_loss` with `best_val_loss`. If `val_loss` decreases, saves the model and resets `patience_counter`. Otherwise, increments `patience_counter`. If `patience_counter` reaches `PATIENCE`, training stops early.

2.8 Model Evaluation Function (evaluate_model)

This function evaluates the model's final performance on the test set after training.

- `model.eval()` and `with torch.no_grad()`: Ensures the model is in evaluation mode and no gradients are computed.
- Iterates through `test_loader`, collecting all predictions (`all_preds`) and true labels (`all_labels`).
- Calculates overall accuracy.
- Uses `sklearn.metrics.confusion_matrix` to generate the confusion matrix (`cm`).
- Computes and prints the accuracy for each class based on the confusion matrix.

2.9 Result Visualization Function (plot_results)

To gain an overview of the learning process and model performance, this function creates important plots:

- **Learning Curves**: Plots `train_loss`, `val_loss`, `train_acc`, `val_acc` against epochs. This plot helps observe learning trends and detect overfitting or underfitting.
- **Confusion Matrix**: Uses `seaborn.heatmap` to visualize the confusion matrix. This plot displays the number of correctly/incorrectly classified samples for each class pair, helping to understand where the model performs well and where it struggles.

Both plots are saved as `.png` files if `plt.show()` does not function.

2.10 Main Function (main)

The `main` function orchestrates the entire training and evaluation process:

- Prints the device being used (GPU or CPU).
- Calls `get_data_loaders()` to prepare the data.
- Initializes the CNN model and moves it to `DEVICE`.
- Sets up the loss function (`criterion`), optimizer (`optimizer`), and learning rate scheduler (`scheduler`).
- Calls `train_model()` to begin the training process.
- After training, reloads the state of the best saved model (`best_model.pth`).
- Calls `evaluate_model()` to assess the model on the test set.
- Calls `plot_results()` to display the result plots.

Chapter 3

Obtained Results

3.1 Observations from Logs and Results

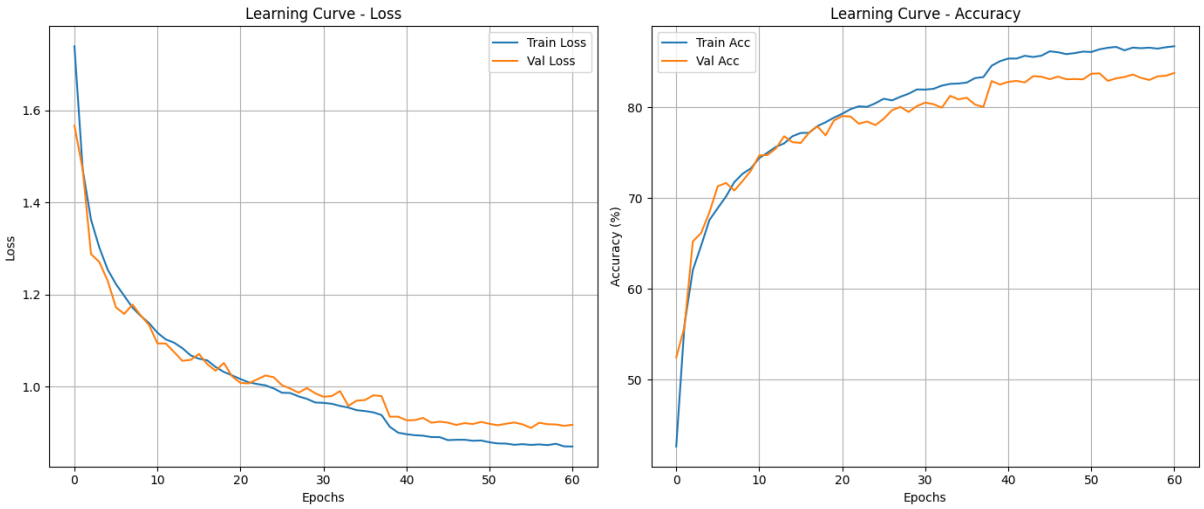

Figure 3.1: Learning Curve Plot

The model was trained on a **GPU (cuda:0)**, leveraging hardware acceleration capabilities.

The training process lasted for **61 epochs** before early stopping (due to `PATIENCE = 5` and insufficient improvement in `val_loss`).

Loss: Both `Train Loss` and `Val Loss` progressively decreased and converged. `Val Loss` reached its lowest point at approximately **0.9104** at epoch 56. Subsequently, it began to slightly increase again before early stopping was triggered.

Accuracy during training/validation: Both `Train Acc` and `Val Acc` gradually increased. `Train Acc` ultimately reached **86.76%** and `Val Acc` reached **83.81%**.

Notable Observation: In the final epochs (e.g., from epoch 51 onwards), `Train Acc` (86.12% - 86.76%) tended to be higher than `Val Acc` (82.94% - 83.81%). This is a good indication that the model is learning features from the training data and generalizing relatively well to the validation set. This difference is normal and often acceptable when using regularization techniques such as `Dropout` and `Data Augmentation`.

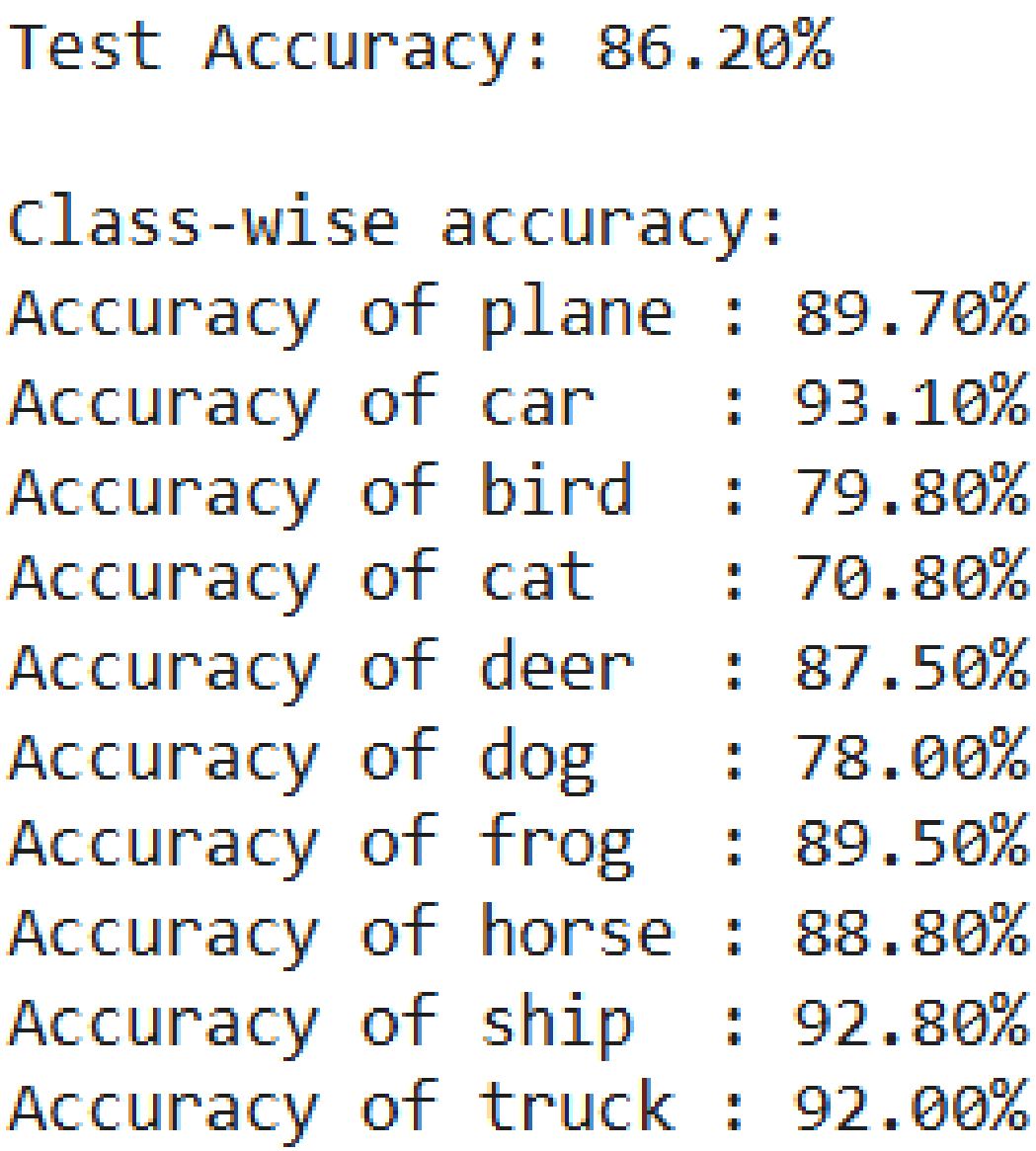

Figure 3.2: Test Results Illustration

3.1.1 Learning Curves

Loss Curve: Both the `Train Loss` and `Validation Loss` decreased sharply in the initial epochs and then converged to a lower level. The `Validation Loss` appeared stable around $0.9 - 0.95$ in the final epochs, indicating that the model learned well and did not severely overfit.

Accuracy Curve: Both `Train Accuracy` and `Validation Accuracy` increased rapidly in the early epochs and then slowed down, converging. `Train Accuracy` consistently remained slightly higher than `Validation Accuracy`, which is a normal sign when regularization techniques are applied.

3.1.2 Confusion Matrix on the Test Set

Overall Test Accuracy: **86.20%**. This is a very good result for the CIFAR-10 classification task with a relatively simple CNN architecture. This accuracy level is close to the

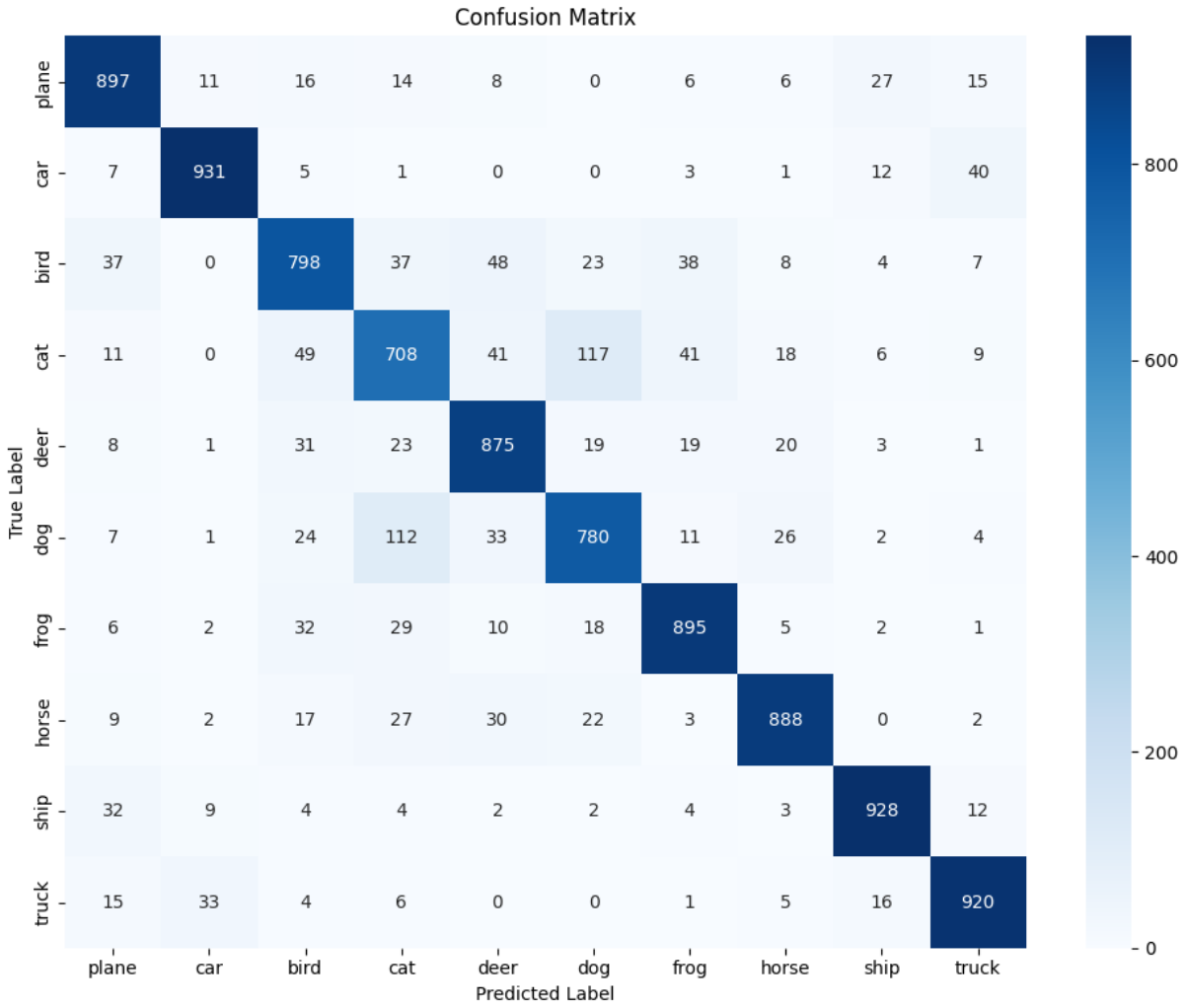

Figure 3.3: Confusion Matrix Plot

final `Train Acc` and `Val Acc`.

Accuracy per Class:

- `plane`: 89.70%
- `car`: 93.10% (**Very high**)
- `bird`: 79.80%
- `cat`: 70.80% (**Lowest**)
- `deer`: 87.50%
- `dog`: 78.00%
- `frog`: 89.50%
- `horse`: 88.80%
- `ship`: 92.80% (**Very high**)
- `truck`: 92.00% (**Very high**)

From the confusion matrix, it is clear that the values on the main diagonal are high, especially for `car`, `ship`, and `truck`. The off-diagonal cells indicate misclassifications between classes. The `cat` and `dog` pair had the highest number of misclassifications (110 cats predicted as dogs and 104 dogs predicted as cats), which is very common in pet image classification problems due to visual similarities.

Chapter 4

Model Evaluation

4.1 Overall Performance

The model achieved very good accuracy on the test set (**86.20%**), demonstrating strong learning and generalization capabilities on unseen data. This is an encouraging result for a CNN model with 3 convolutional layers on CIFAR-10.

- Both `Train Loss` and `Val Loss` decreased steadily, indicating an effective optimization process.
- Both `Train Acc` and `Val Acc` grew in parallel and converged, showing that the model learned good features and generalized effectively.

4.2 The Phenomenon of Train Acc > Val Acc (and Explanation)

The accuracy on the training set (`Train Acc`) tended to be higher than the accuracy on the validation set (`Val Acc`). This is a normal and expected phenomenon in deep learning training, especially when using regularization techniques:

- **Dropout**: During training (`model.train()`), neurons are randomly deactivated, making the network weaker and harder to achieve the highest accuracy on the training set. Conversely, during evaluation (`model.eval()`), `Dropout` is turned off, allowing the entire network to operate simultaneously, helping the model achieve higher potential performance on the validation/test set.
- **Batch Normalization**: During training, `Batch Normalization` uses batch statistics (mean and variance) of the current batch. During evaluation, it uses global average statistics (`running means/variances`) computed during training, providing more stable estimates and often leading to better performance.
- **Data Augmentation**: Transformations such as `RandomCrop`, `RandomHorizontalFlip`, `RandomErasing` make the training data more diverse and challenging. The model has to learn from augmented versions of images, whereas validation/test data does not undergo these random transformations, which might make them "easier" for a well-trained model.

- **Label Smoothing**: Reduces the model's confidence in a single label during training, which can lead to slightly higher `Train Loss` and indirectly affect `Train Acc` compared to `Val Acc` which is not influenced by this technique.

In summary, the phenomenon of `Train Acc` being slightly higher than `Val Acc` is a good sign that regularization techniques have worked effectively, preventing the model from overfitting to the training data while maintaining good generalization capabilities.

4.3 Overfitting/Underfitting Potential

- **No clear signs of Overfitting**: Although `Train Acc` is higher than `Val Acc`, the difference is not excessively large, and `Val Loss` continued to decrease until early stopping was triggered, indicating that the model is still generalizing well. The Learning Curves (loss and accuracy) show reasonable convergence.
- **No Underfitting**: The accuracy on both the training and validation sets is high and stable, demonstrating that the model is sufficiently complex to learn the necessary features from the data.

4.4 Class-wise Accuracy

Analyzing the accuracy for each class provides a deeper insight into the model's performance:

- Classes such as `car` (**93.10%**), `ship` (**92.80%**), `truck` (**92.00%**), `plane` (**89.70%**), `frog` (**89.50%**), `horse` (**88.80%**) are classified very well. This indicates that the model has learned clear and distinguishable features for these objects, especially vehicles.
- The `bird` (**79.80%**) and `dog` (**78.00%**) classes have lower accuracy.
- The `cat` (**70.80%**) class has the lowest accuracy. The confusion matrix confirms that `cat` and `dog` are the two most frequently confused classes (214 cases), which is very common in pet image classification problems due to visual similarities.

4.5 Evaluation of Hyperparameters and Techniques

- **3-layer CNN Architecture**: Sufficiently powerful to learn complex features on CIFAR-10.
- **Data Augmentation**: Very important for increasing data diversity and improving generalization.
- **Batch Normalization**: Helps stabilize the training process and accelerate convergence.
- **Dropout, Label Smoothing, Weight Decay**: These regularization techniques have been effective in preventing overfitting and helping the model achieve high overall performance on the test set.

- **ReduceLROnPlateau Scheduler**: Effective in adjusting the `LR`, helping the model escape local minima and find a better optimum. The `LR` decreased from 1.0e-03 to 1.0e-04 and finally to 1.0e-06.
- **Early Stopping (PATIENCE=5)**: Helped stop the training process at epoch 61, saving time and preventing potential overfitting without needing to run all 300 epochs.

Chapter 5

Conclusion

This assignment successfully built, trained, and evaluated a 3-layer CNN for image classification on the CIFAR-10 dataset using PyTorch. The model achieved an accuracy of **86.20%** on the test set, which is a very good result. Techniques such as `Data Augmentation`, `Batch Normalization`, `Dropout`, `Label Smoothing`, `Weight Decay`, `Learning Rate Scheduling`, and `Early Stopping` were effectively applied to improve performance, prevent overfitting, and ensure good generalization capabilities of the model.

Although the `cat` class remains the biggest challenge for the model due to high visual similarity with `dog`, the overall results indicate that a robust image classification model has been built. To further improve performance, consider:

- Deeper analysis of the confusion matrix to understand the causes of misclassifications between classes.
- Experimenting with more advanced CNN architectures (e.g., `ResNet`, `DenseNet`, `VGG`).
- Finer tuning of `hyperparameters`, especially `DROPOUT_RATE`, `LABEL_SMOOTHING`, `WEIGHT_DECAY`, and scheduler parameters.
- Using more advanced `Data Augmentation` techniques or methods for handling class imbalance if necessary.