CIFAR-10 Classification

Amirhossein Safari

1. Introduction

In this report, we detail the implementation of a deep learning model for CIFAR-10 classification and discuss the various generalization techniques applied to ensure robust performance on unseen data. The model, designed with fewer than 400,000 parameters, leverages advanced regularization and optimization strategies to prevent overfitting while maintaining high accuracy.

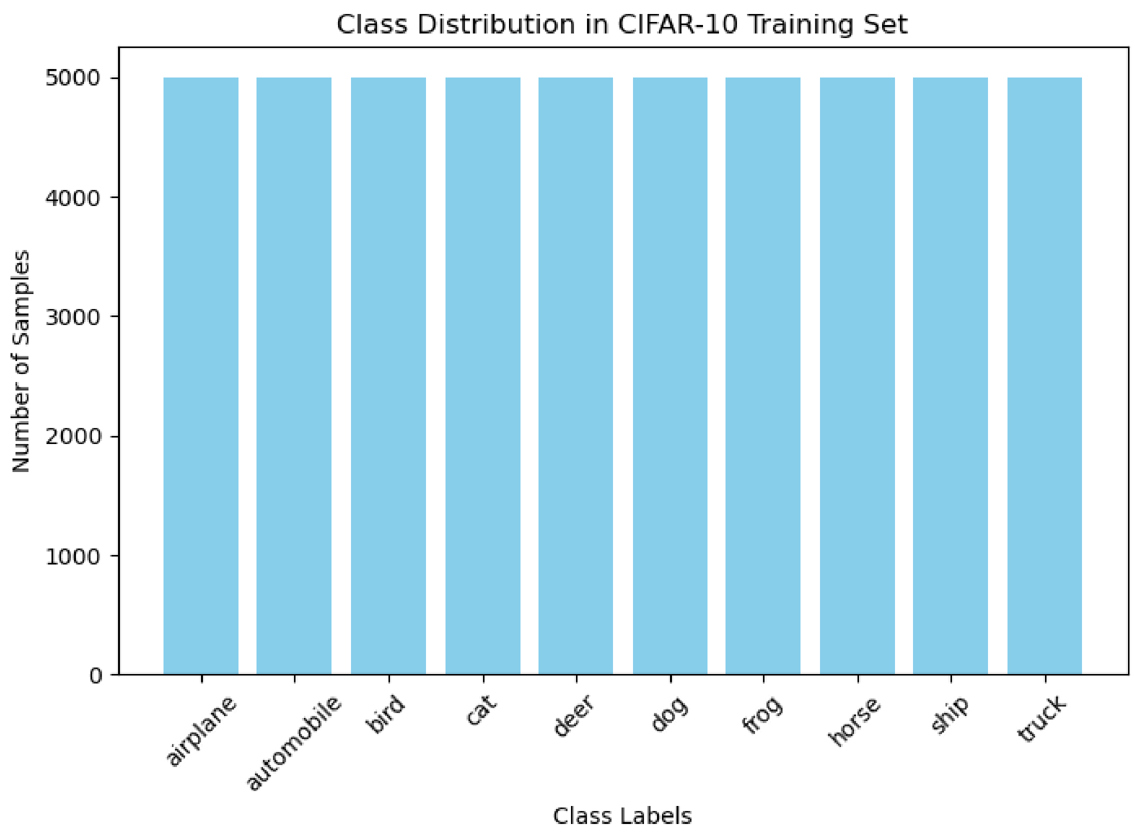
2. Data Description

2.1 Dataset Overview

- **Dataset**:CIFAR-10
  CIFAR-10 is a widely used benchmark dataset in computer vision that consists of 60,000 32x32 color images categorized into 10 classes. The dataset is split into 50,000 training images and 10,000 test images.

2.2 Data Characteristics

- **Image Dimensions**: Each image is 32x32 pixels with 3 color channels.
- **Classes**: The classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.
- **Variability**: Images exhibit variations in background, lighting, and object orientation, which makes the dataset an excellent candidate for evaluating the model's generalization capabilities.

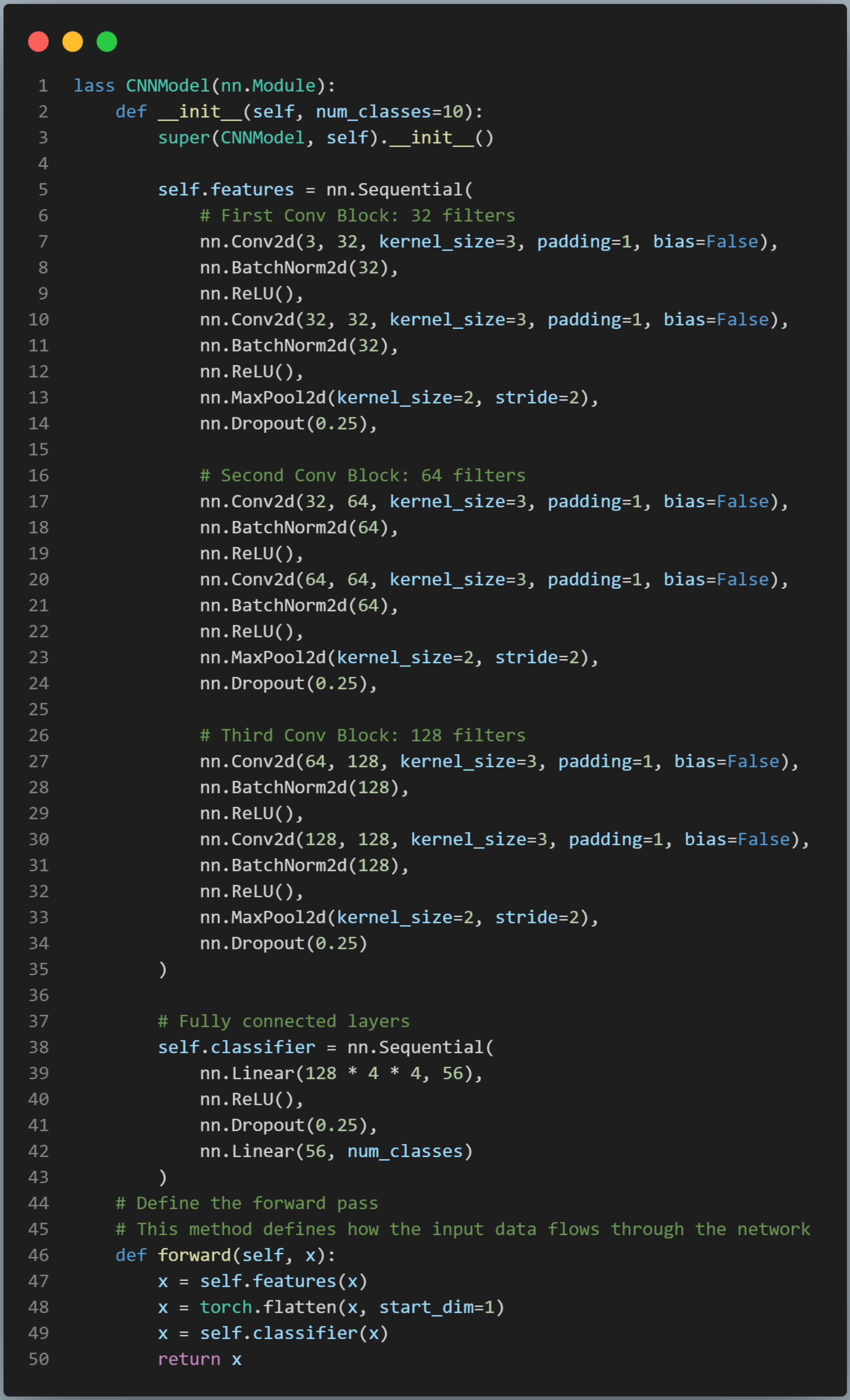


3. Implementation Steps

3.1 Model Architecture

The model is a Convolutional Neural Network (CNN) designed specifically for CIFAR-10, consisting of the following components:

- **Feature Extraction Layers**:
  The network contains several convolutional blocks with increasing numbers of filters:
  - **Block 1**: Two convolutional layers with 32 filters each, followed by batch normalization, ReLU activations, max-pooling, and dropout.
  - **Block 2**: Two convolutional layers with 64 filters each, along with batch normalization, ReLU activations, max-pooling, and dropout.
  - **Block 3**: Two convolutional layers with 128 filters each, followed by batch normalization, ReLU activations, max-pooling, and dropout.
- **Classifier**:
  The classifier consists of fully connected layers that convert the feature maps into class probabilities. The design intentionally keeps the overall parameter count under 400,000 while ensuring robust performance

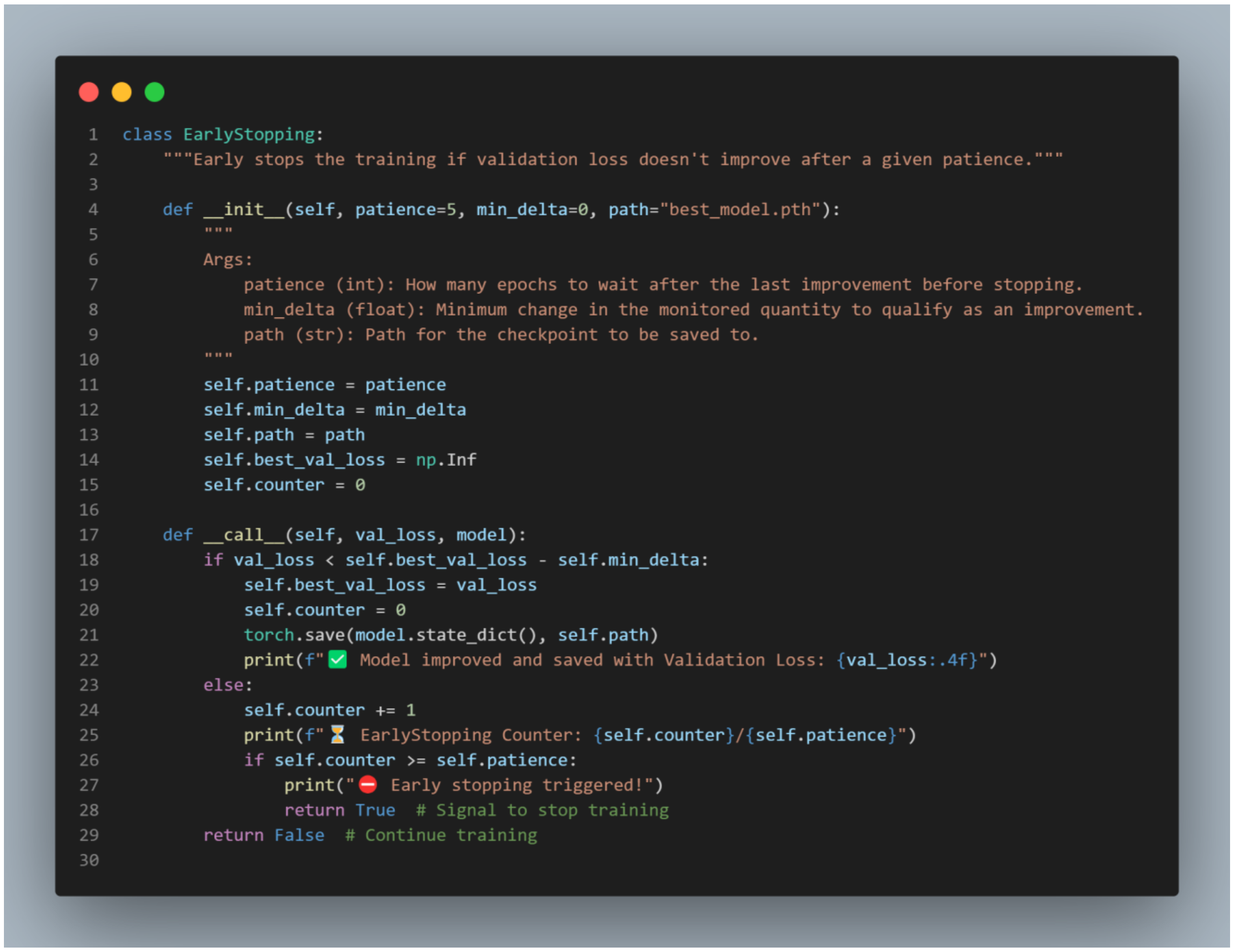


```python
lass CNNModel(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNModel, self).__init__()

        self.features = nn.Sequential(
            # First Conv Block: 32 filters
            nn.Conv2d(3, 32, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout(0.25),

            # Second Conv Block: 64 filters
            nn.Conv2d(32, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout(0.25),

            # Third Conv Block: 128 filters
            nn.Conv2d(64, 128, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout(0.25)
        )

        # Fully connected layers
        self.classifier = nn.Sequential(
            nn.Linear(128 * 4 * 4, 56),
            nn.ReLU(),
            nn.Dropout(0.25),
            nn.Linear(56, num_classes)
        )
    # Define the forward pass
    # This method defines how the input data flows through the network
    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, start_dim=1)
        x = self.classifier(x)
        return x
```

3.2 Training Process

The training process incorporates several critical strategies:

- **Loss Function and Optimizer**:
  The model is trained using a cross-entropy loss function with label smoothing to prevent overconfidence. The Adam optimizer, combined with weight decay (L2 regularization), helps in promoting simpler and more generalizable weights.
- **Learning Rate Scheduling**:
  A learning rate scheduler (StepLR) is employed to dynamically adjust the learning rate every 10 epochs, which aids in smoother convergence during training.
- **Early Stopping**:
  An early stopping mechanism monitors the validation loss. If the validation loss does not improve for a specified number of consecutive epochs (patience), the training process is halted to prevent overfitting.



```python
class EarlyStopping:
    """Early stops the training if validation loss doesn't improve after a given patience."""

    def __init__(self, patience=5, min_delta=0, path="best_model.pth"):
        """
        Args:
            patience (int): How many epochs to wait after the last improvement before stopping.
            min_delta (float): Minimum change in the monitored quantity to qualify as an improvement.
            path (str): Path for the checkpoint to be saved to.
        """
        self.patience = patience
        self.min_delta = min_delta
        self.path = path
        self.best_val_loss = np.Inf
        self.counter = 0

    def __call__(self, val_loss, model):
        if val_loss < self.best_val_loss - self.min_delta:
            self.best_val_loss = val_loss
            self.counter = 0
            torch.save(model.state_dict(), self.path)
            print(f"✅ Model improved and saved with Validation Loss: {val_loss:.4f}")
        else:
            self.counter += 1
            print(f"⏳ EarlyStopping Counter: {self.counter}/{self.patience}")
            if self.counter >= self.patience:
                print("⛔ Early stopping triggered!")
                return True  # Signal to stop training
        return False  # Continue training

```

3.3 Training Loop

The training loop consists of the following steps:

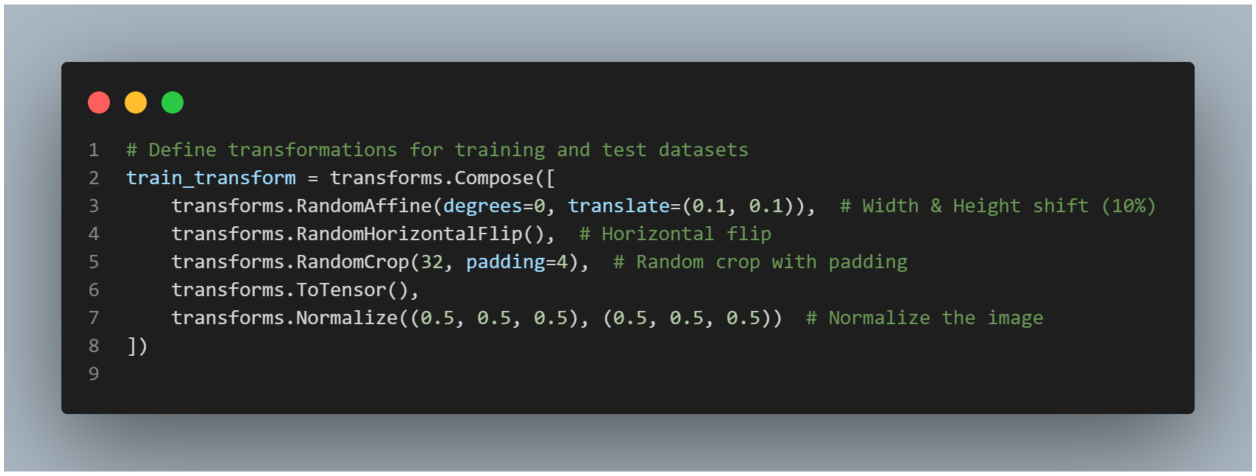
1. **Training Phase**:
   - The model is set to training mode.
   - The network processes batches of training images, computes the loss, performs backpropagation, and updates the weights.
   - Training loss and accuracy are calculated and tracked.
2. **Validation Phase**:
   - The model is switched to evaluation mode.
   - Validation loss and accuracy are computed without updating model weights.
3. **Scheduler and Early Stopping**:
   - The learning rate is updated using the scheduler.
   - The early stopping criterion is checked to determine if training should be halted.
4. **Metrics Visualization**:
   - Loss and accuracy curves are plotted to visualize the model's performance over epochs.

4. Generalization Techniques

The following regularization methods are applied to enhance model generalization:

4.1 Data-Level Techniques

- **Data Augmentation**: Enhances the diversity of the training set by applying random transformations, which helps the model learn more invariant features.



```python
# Define transformations for training and test datasets
train_transform = transforms.Compose([
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),  # Width & Height shift (10%)
    transforms.RandomHorizontalFlip(),  # Horizontal flip
    transforms.RandomCrop(32, padding=4),  # Random crop with padding
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize the image
])

```

- **Label Smoothing**: Mitigates overconfidence by adjusting the target labels, thereby improving model robustness.

4.2 Architecture-Level Techniques

- **Batch Normalization**: Normalizes the inputs of each layer to speed up training and stabilize the learning process.
- **Dropout**: Randomly deactivates neurons during training to prevent the model from becoming overly reliant on specific features.
- **Weight Decay (L2 Regularization)**: Adds a regularization term to the loss function to penalize large weights, encouraging the development of simpler and more generalizable models.

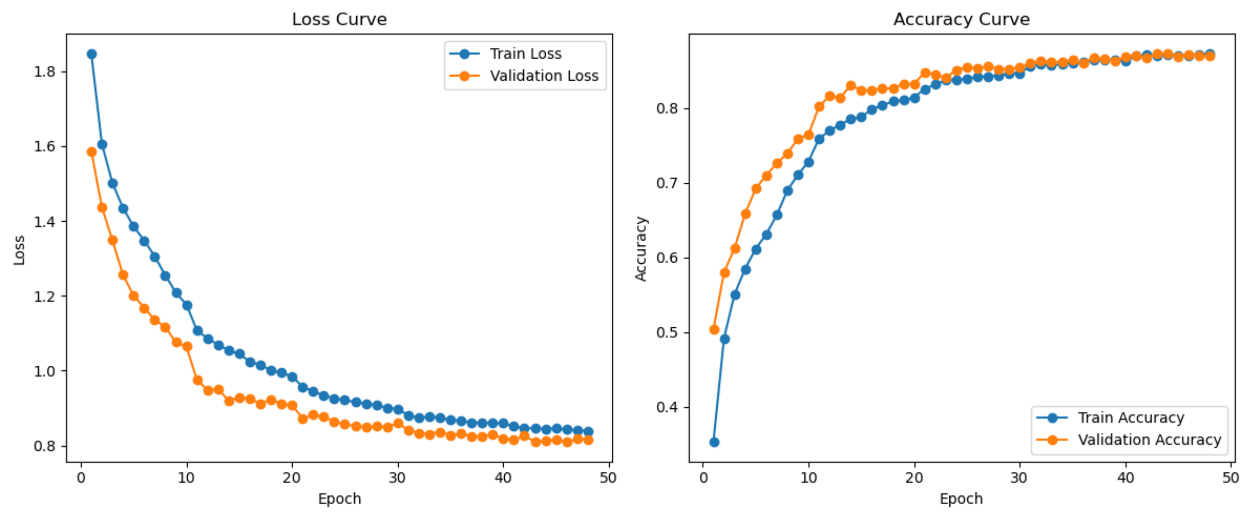
4.3 Optimization Strategies

- **Learning Rate Scheduling**: Dynamically adjusts the learning rate to facilitate efficient convergence.
- **Early Stopping**: Monitors validation loss to terminate training when further improvements are unlikely, thus preventing overfitting.

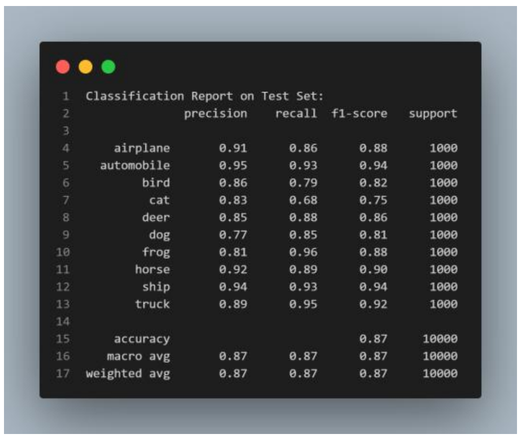
5. Results

The performance of the model was evaluated over multiple epochs, and key observations are as follows:

- **Loss Curves**:
  Both training and validation loss curves show a steady decline, indicating effective learning and stability. The close alignment of these curves suggests that the generalization techniques are successfully mitigating overfitting.
- **Accuracy Trends**:
  The accuracy curves for both training and validation data exhibit a positive trend, with the model achieving its best performance just before early stopping was triggered. This demonstrates that the model is generalizing well despite its compact architecture.

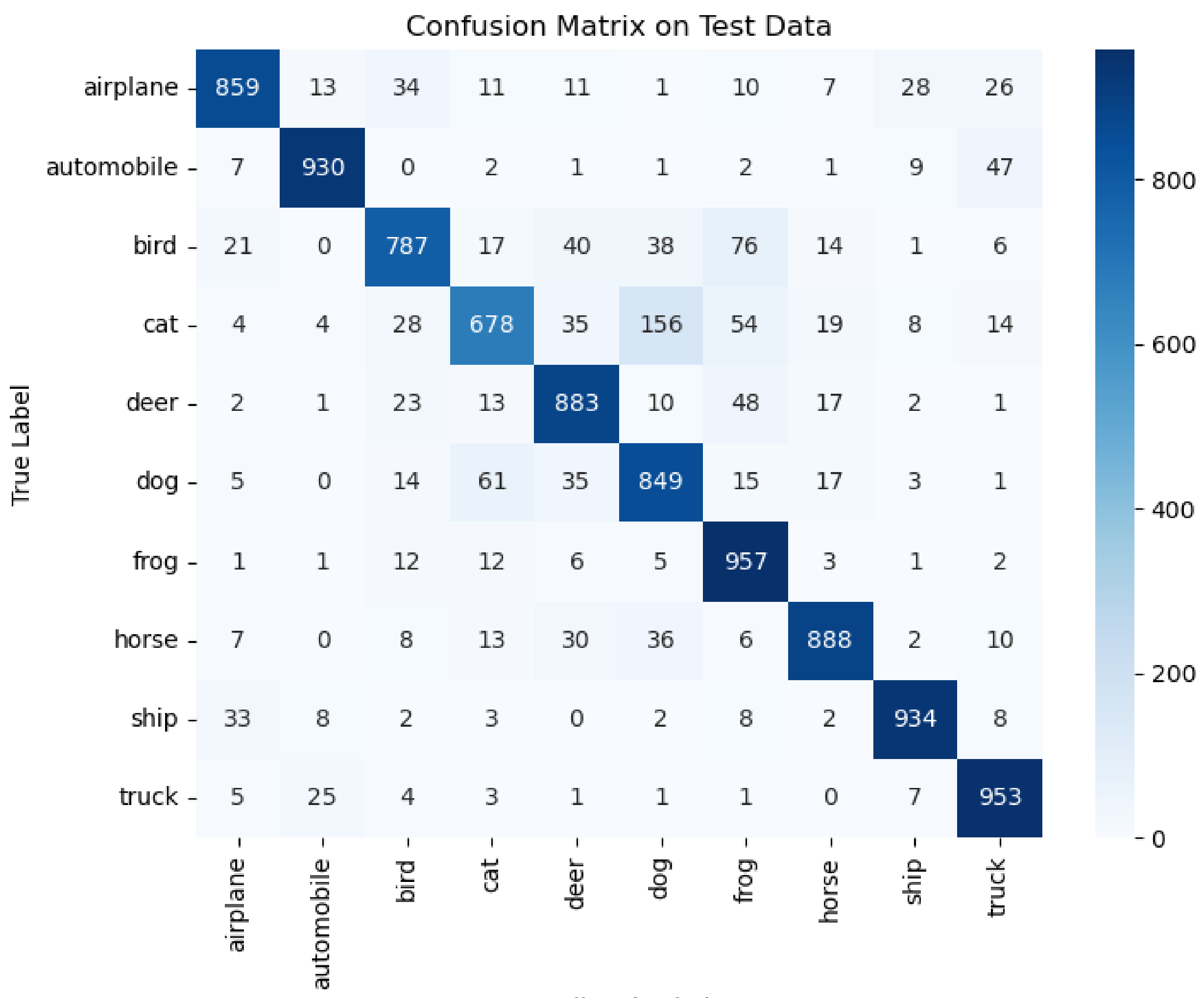


Loss and Accuracy Curve



```
Classification Report on Test Set:
              precision    recall  f1-score   support

    airplane       0.91      0.86      0.88      1000
  automobile       0.95      0.93      0.94      1000
        bird       0.86      0.79      0.82      1000
         cat       0.83      0.68      0.75      1000
        deer       0.85      0.88      0.86      1000
         dog       0.77      0.85      0.81      1000
        frog       0.81      0.96      0.88      1000
       horse       0.92      0.89      0.90      1000
        ship       0.94      0.93      0.94      1000
       truck       0.89      0.95      0.92      1000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

Classification Report on Test Set



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	859	13	34	11	11	1	10	7	28	26
automobile	7	930	0	2	1	1	2	1	9	47
bird	21	0	787	17	40	38	76	14	1	6
cat	4	4	28	678	35	156	54	19	8	14
deer	2	1	23	13	883	10	48	17	2	1
dog	5	0	14	61	35	849	15	17	3	1
frog	1	1	12	12	6	5	957	3	1	2
horse	7	0	8	13	30	36	6	888	2	10
ship	33	8	2	3	0	2	8	2	934	8
truck	5	25	4	3	1	1	1	0	7	953

Confusion Matrix on Test Data