st125981_CV_A2_Task_1

November 1, 2025

0.1 CV Assignment 2 - Task 1

Name: Muhammad Fahad Waqar Student No: st125981

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn.metrics import confusion_matrix
import pandas as pd
import time
from copy import deepcopy
```

[2]:
```
# Configuration
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {DEVICE}")

# Hyperparameters
BATCH_SIZE = 128
NUM_EPOCHS = 20
LR_ADAM = 0.001
LR_SGD = 0.01
MOMENTUM_SGD = 0.9
```

```
Using device: cuda
```

[3]:
```
# Data Loading
CLASS_NAMES = ('plane', 'car', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck')

# Normalization stats for CIFAR-10
cifar_mean = (0.4914, 0.4822, 0.4465)
```

```
cifar_std = (0.2470, 0.2435, 0.2616)

# Training transformations
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(cifar_mean, cifar_std)
])

# Testing transformations
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(cifar_mean, cifar_std)
])

# Loading datasets
try:
    train_dataset = torchvision.datasets.CIFAR10(
        root='./data', train=True, download=True, transform=transform_train)

    test_dataset = torchvision.datasets.CIFAR10(
        root='./data', train=False, download=True, transform=transform_test)

    # Create DataLoaders
    train_loader = DataLoader(
        train_dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)

    test_loader = DataLoader(
        test_dataset, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)

    print(f"Loaded CIFAR-10 dataset.")
    print(f"Training samples: {len(train_dataset)}")
    print(f"Test samples: {len(test_dataset)}")

except Exception as e:
    print(f"Error loading CIFAR-10 dataset. Make sure you have an internet␣
 ↪connection. Error: {e}")
    train_loader = None
    test_loader = None
```

```
Files already downloaded and verified
Files already downloaded and verified
Loaded CIFAR-10 dataset.
Training samples: 50000
Test samples: 10000
```

[4]:
```
# Model Definitions
# Model 1: Simple CNN
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()

        # Convolutional Layer 1
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Convolutional Layer 2
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(32 * 8 * 8, 256)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Model 2: ResNet-18
def get_resnet_model(num_classes=10):
    model = torchvision.models.resnet18(weights=torchvision.models.
 ↪ResNet18_Weights.DEFAULT)

    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, num_classes)

    return model
```

[5]:
```
# Training and Evaluation Functions
def train_one_epoch(model, train_loader, optimizer, criterion, device):
    model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)
```

```
        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        total_samples += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

    epoch_loss = running_loss / total_samples
    epoch_acc = 100.0 * correct_predictions / total_samples
    return epoch_loss, epoch_acc

def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    all_labels = []
    all_predictions = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs.data, 1)
            total_samples += labels.size(0)
            correct_predictions += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    epoch_loss = running_loss / total_samples
    epoch_acc = 100.0 * correct_predictions / total_samples
    return epoch_loss, epoch_acc, all_labels, all_predictions
```

[6]:
```
# Learning Rate Scheduler Configurations
scheduler_configs = [
    {
        'name': 'None',
        'builder': lambda optimizer: None  # No scheduler
```

```
    },
    {
        'name': 'StepLR',
        'builder': lambda optimizer: optim.lr_scheduler.StepLR(
            optimizer, step_size=7, gamma=0.1
        )
    },
    {
        'name': 'ReduceLROnPlateau',
        'builder': lambda optimizer: optim.lr_scheduler.ReduceLROnPlateau(
            optimizer, mode='min', factor=0.1, patience=5, verbose=True
        )
    }
]

# Updated Training Function with Scheduler Support
def train_one_epoch_with_scheduler(model, train_loader, optimizer, criterion,␣
 ↪device):
    """Training function (same as before, no changes needed)"""
    model.train()
    running_loss = 0.0
    correct_predictions = 0
    total_samples = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        total_samples += labels.size(0)
        correct_predictions += (predicted == labels).sum().item()

    epoch_loss = running_loss / total_samples
    epoch_acc = 100.0 * correct_predictions / total_samples
    return epoch_loss, epoch_acc
```

[7]:
```
# Visualization Functions
def plot_history(history, title):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

    # Accuracy
```

```
    ax1.plot(history['train_acc'], label='Train Accuracy')
    ax1.plot(history['val_acc'], label='Validation Accuracy')
    ax1.set_title(f'Accuracy\n{title}')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy (%)')
    ax1.legend()
    ax1.grid(True)

    # Loss
    ax2.plot(history['train_loss'], label='Train Loss')
    ax2.plot(history['val_loss'], label='Validation Loss')
    ax2.set_title(f'Loss\n{title}')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True)

    plt.suptitle(f'Training History: {title}', fontsize=16)
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()


def plot_confusion_matrix(all_labels, all_preds, class_names, title):
    cm = confusion_matrix(all_labels, all_preds)
    df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)

    plt.figure(figsize=(10, 8))
    sn.heatmap(df_cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix: {title}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()


def plot_misclassified(model, test_loader, class_names, device, title,␣
 ↪num_images=10):
    model.eval()
    misclassified_images = []
    misclassified_labels = []
    misclassified_preds = []

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
```

```
            # Find mismatches
            mismatches = (predicted != labels)
            for i in range(images.size(0)):
                if mismatches[i] and len(misclassified_images) < num_images:
                    img = images[i].cpu()
                    img = img * torch.tensor(cifar_std).view(3, 1, 1) + torch.
 ↪tensor(cifar_mean).view(3, 1, 1)
                    img = transforms.ToPILImage()(img)

                    misclassified_images.append(img)
                    misclassified_labels.append(class_names[labels[i]])
                    misclassified_preds.append(class_names[predicted[i]])

    # Plot the images
    if not misclassified_images:
        print(f"[{title}] No misclassified images found (or failed to find␣
 ↪{num_images}).")
        return

    fig, axes = plt.subplots(1, len(misclassified_images), figsize=(20, 4))
    if len(misclassified_images) == 1:
        axes = [axes]

    for i, ax in enumerate(axes):
        ax.imshow(misclassified_images[i])
        ax.set_title(f"True: {misclassified_labels[i]}\nPred:␣
 ↪{misclassified_preds[i]}")
        ax.axis('off')

    plt.suptitle(f'Example Misclassified Images: {title}')
    plt.tight_layout()
    plt.show()


def plot_activation_maps(model, layers_to_plot, test_loader, device, title,␣
 ↪num_images=5):
    model.eval()
    images, _ = next(iter(test_loader))
    images = images.to(device)

    activations = {}
    handles = []

    def get_activation(name):
        def hook(model, input, output):
            activations[name] = output.detach().cpu()
        return hook
```

```
    for layer_name, layer in layers_to_plot.items():
        handles.append(layer.register_forward_hook(get_activation(layer_name)))

    with torch.no_grad():
        _ = model(images)

    for handle in handles:
        handle.remove()

    # Plot original images
    fig, axes = plt.subplots(num_images, 1, figsize=(3, 2 * num_images))
    if num_images == 1: axes = [axes]
    plt.suptitle(f'Original Images\n{title}', x=0.5, y=1.02, fontsize=14)
    for i in range(num_images):
        img = images[i].cpu()
        img = img * torch.tensor(cifar_std).view(3, 1, 1) + torch.
↪tensor(cifar_mean).view(3, 1, 1)
        img = img.permute(1, 2, 0)
        img = np.clip(img, 0, 1)
        axes[i].imshow(img)
        axes[i].axis('off')
    plt.show()

    # Plot activation maps for each layer
    for layer_name, activation in activations.items():
        num_features = activation.size(1)

        # Plotting the first 8 features
        features_to_plot = min(num_features, 8)

        fig, axes = plt.subplots(num_images, features_to_plot,␣
↪figsize=(features_to_plot * 2, num_images * 2))

        if num_images == 1:
            axes = axes.reshape(1, -1)
        if features_to_plot == 1:
            axes = axes.reshape(-1, 1)

        for i in range(num_images):
            for j in range(features_to_plot):
                ax = axes[i, j]
                ax.imshow(activation[i, j], cmap='viridis')
                ax.axis('off')
                if i == 0:
                    ax.set_title(f'Filter {j+1}')
```

```
        plt.suptitle(f'Activation Maps: {layer_name}\n{title}', fontsize=16)
        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
        plt.show()
```

[8]:
```
# Experiment
model_configs = [
    {'name': 'SimpleCNN', 'builder': SimpleCNN},
    {'name': 'ResNet-18', 'builder': get_resnet_model}
]

# Define the 4 optimizer configurations
optimizer_configs = [
    {
        'name': 'Adam',
        'builder': lambda params: optim.Adam(params, lr=LR_ADAM)
    },
    {
        'name': 'SGD_Vanilla',
        'builder': lambda params: optim.SGD(params, lr=LR_SGD)
    },
    {
        'name': 'SGD_Momentum',
        'builder': lambda params: optim.SGD(params, lr=LR_SGD,␣
 ↪momentum=MOMENTUM_SGD)
    },
    {
        'name': 'SGD_Nesterov',
        'builder': lambda params: optim.SGD(params, lr=LR_SGD,␣
 ↪momentum=MOMENTUM_SGD, nesterov=True)
    }
]

# Loss function
criterion = nn.CrossEntropyLoss()

# Dictionary to store all results
experiment_results = {}

# Check if data loaders are available
if train_loader is None or test_loader is None:
    print("FATAL: Data loaders were not initialized. Stopping experiment.")
else:
    # --- Main Training Loop ---

    for model_config in model_configs:
        for optimizer_config in optimizer_configs:
```

```
            experiment_name =␣
↪f"{model_config['name']}_{optimizer_config['name']}"
            print(f"STARTING EXPERIMENT: {experiment_name}")

            # 1. Initialize model and optimizer
            model = model_config['builder']().to(DEVICE)
            optimizer = optimizer_config['builder'](model.parameters())

            # Store history for this run
            history = {
                'train_loss': [], 'train_acc': [],
                'val_loss': [], 'val_acc': []
            }

            start_time = time.time()

            # 2. Run training epochs
            for epoch in range(NUM_EPOCHS):

                # Train
                train_loss, train_acc = train_one_epoch(
                    model, train_loader, optimizer, criterion, DEVICE)

                # Validate
                val_loss, val_acc, _, _ = evaluate_model(
                    model, test_loader, criterion, DEVICE)

                # Log results
                history['train_loss'].append(train_loss)
                history['train_acc'].append(train_acc)
                history['val_loss'].append(val_loss)
                history['val_acc'].append(val_acc)

                print(f"  Epoch [{epoch+1:02d}/{NUM_EPOCHS}] | "
                      f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.
↪2f}% | "
                      f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%")

            end_time = time.time()
            print(f"FINISHED EXPERIMENT: {experiment_name}")
            print(f"   Total Time: {end_time - start_time:.2f} seconds")

            # 3. Store results (model state and history)
            experiment_results[experiment_name] = {
                'model_state': deepcopy(model.state_dict()), # Save a copy of␣
↪the weights
                'model_config': model_config,
```

```
            'history': history,
            'time': end_time - start_time
        }

        # Clean up memory
        del model
        del optimizer
        if DEVICE == 'cuda':
            torch.cuda.empty_cache()

print("All 8 experiments completed!")
```

```
STARTING EXPERIMENT: SimpleCNN_Adam
  Epoch [01/20] | Train Loss: 1.6083, Train Acc: 41.47% | Val Loss: 1.2970, Val
Acc: 54.30%
  Epoch [02/20] | Train Loss: 1.3215, Train Acc: 52.60% | Val Loss: 1.1343, Val
Acc: 59.13%
  Epoch [03/20] | Train Loss: 1.1899, Train Acc: 57.52% | Val Loss: 1.0625, Val
Acc: 62.96%
  Epoch [04/20] | Train Loss: 1.0991, Train Acc: 60.83% | Val Loss: 0.9978, Val
Acc: 64.49%
  Epoch [05/20] | Train Loss: 1.0382, Train Acc: 63.19% | Val Loss: 0.9326, Val
Acc: 67.43%
  Epoch [06/20] | Train Loss: 0.9912, Train Acc: 64.81% | Val Loss: 0.9000, Val
Acc: 68.74%
  Epoch [07/20] | Train Loss: 0.9488, Train Acc: 66.74% | Val Loss: 0.9040, Val
Acc: 68.45%
  Epoch [08/20] | Train Loss: 0.9131, Train Acc: 67.75% | Val Loss: 0.8395, Val
Acc: 70.45%
  Epoch [09/20] | Train Loss: 0.8872, Train Acc: 68.68% | Val Loss: 0.8355, Val
Acc: 70.45%
  Epoch [10/20] | Train Loss: 0.8615, Train Acc: 69.85% | Val Loss: 0.8295, Val
Acc: 71.06%
  Epoch [11/20] | Train Loss: 0.8362, Train Acc: 70.57% | Val Loss: 0.7772, Val
Acc: 73.35%
  Epoch [12/20] | Train Loss: 0.8192, Train Acc: 71.35% | Val Loss: 0.7939, Val
Acc: 72.38%
  Epoch [13/20] | Train Loss: 0.8039, Train Acc: 71.76% | Val Loss: 0.7580, Val
Acc: 73.91%
  Epoch [14/20] | Train Loss: 0.7855, Train Acc: 72.43% | Val Loss: 0.7628, Val
Acc: 73.49%
  Epoch [15/20] | Train Loss: 0.7713, Train Acc: 72.96% | Val Loss: 0.7731, Val
Acc: 73.25%
  Epoch [16/20] | Train Loss: 0.7600, Train Acc: 73.26% | Val Loss: 0.7438, Val
Acc: 73.94%
  Epoch [17/20] | Train Loss: 0.7488, Train Acc: 73.73% | Val Loss: 0.7295, Val
Acc: 74.85%
  Epoch [18/20] | Train Loss: 0.7331, Train Acc: 74.19% | Val Loss: 0.7107, Val
```

```
Acc: 75.52%
  Epoch [19/20] | Train Loss: 0.7232, Train Acc: 74.65% | Val Loss: 0.7422, Val
Acc: 73.91%
  Epoch [20/20] | Train Loss: 0.7166, Train Acc: 75.01% | Val Loss: 0.7047, Val
Acc: 75.77%
FINISHED EXPERIMENT: SimpleCNN_Adam
   Total Time: 542.31 seconds
STARTING EXPERIMENT: SimpleCNN_SGD_Vanilla
  Epoch [01/20] | Train Loss: 2.1298, Train Acc: 23.89% | Val Loss: 1.9605, Val
Acc: 29.54%
  Epoch [02/20] | Train Loss: 1.8860, Train Acc: 32.30% | Val Loss: 1.7888, Val
Acc: 35.74%
  Epoch [03/20] | Train Loss: 1.7511, Train Acc: 36.68% | Val Loss: 1.6521, Val
Acc: 40.21%
  Epoch [04/20] | Train Loss: 1.6324, Train Acc: 40.67% | Val Loss: 1.5316, Val
Acc: 43.60%
  Epoch [05/20] | Train Loss: 1.5469, Train Acc: 43.83% | Val Loss: 1.4659, Val
Acc: 47.10%
  Epoch [06/20] | Train Loss: 1.4903, Train Acc: 46.20% | Val Loss: 1.4199, Val
Acc: 48.57%
  Epoch [07/20] | Train Loss: 1.4460, Train Acc: 48.06% | Val Loss: 1.3976, Val
Acc: 49.79%
  Epoch [08/20] | Train Loss: 1.4113, Train Acc: 49.39% | Val Loss: 1.3302, Val
Acc: 52.96%
  Epoch [09/20] | Train Loss: 1.3848, Train Acc: 50.31% | Val Loss: 1.3028, Val
Acc: 53.85%
  Epoch [10/20] | Train Loss: 1.3545, Train Acc: 51.69% | Val Loss: 1.2927, Val
Acc: 54.44%
  Epoch [11/20] | Train Loss: 1.3295, Train Acc: 52.40% | Val Loss: 1.2553, Val
Acc: 55.37%
  Epoch [12/20] | Train Loss: 1.3038, Train Acc: 53.45% | Val Loss: 1.2352, Val
Acc: 56.36%
  Epoch [13/20] | Train Loss: 1.2822, Train Acc: 54.33% | Val Loss: 1.2086, Val
Acc: 57.60%
  Epoch [14/20] | Train Loss: 1.2583, Train Acc: 55.13% | Val Loss: 1.1931, Val
Acc: 57.89%
  Epoch [15/20] | Train Loss: 1.2374, Train Acc: 56.14% | Val Loss: 1.1348, Val
Acc: 59.88%
  Epoch [16/20] | Train Loss: 1.2181, Train Acc: 56.63% | Val Loss: 1.1486, Val
Acc: 59.73%
  Epoch [17/20] | Train Loss: 1.1936, Train Acc: 57.72% | Val Loss: 1.1138, Val
Acc: 61.01%
  Epoch [18/20] | Train Loss: 1.1753, Train Acc: 58.45% | Val Loss: 1.0974, Val
Acc: 61.79%
  Epoch [19/20] | Train Loss: 1.1545, Train Acc: 59.44% | Val Loss: 1.1063, Val
Acc: 60.97%
  Epoch [20/20] | Train Loss: 1.1400, Train Acc: 59.80% | Val Loss: 1.0714, Val
Acc: 61.84%
```

```
FINISHED EXPERIMENT: SimpleCNN_SGD_Vanilla
   Total Time: 472.41 seconds
STARTING EXPERIMENT: SimpleCNN_SGD_Momentum
  Epoch [01/20] | Train Loss: 1.7873, Train Acc: 34.54% | Val Loss: 1.4214, Val
Acc: 48.12%
  Epoch [02/20] | Train Loss: 1.4230, Train Acc: 48.09% | Val Loss: 1.2711, Val
Acc: 54.81%
  Epoch [03/20] | Train Loss: 1.2822, Train Acc: 53.63% | Val Loss: 1.1620, Val
Acc: 59.25%
  Epoch [04/20] | Train Loss: 1.1677, Train Acc: 58.10% | Val Loss: 1.0345, Val
Acc: 63.28%
  Epoch [05/20] | Train Loss: 1.0913, Train Acc: 61.05% | Val Loss: 0.9968, Val
Acc: 64.65%
  Epoch [06/20] | Train Loss: 1.0337, Train Acc: 63.55% | Val Loss: 0.9420, Val
Acc: 67.16%
  Epoch [07/20] | Train Loss: 0.9859, Train Acc: 65.18% | Val Loss: 0.9005, Val
Acc: 68.78%
  Epoch [08/20] | Train Loss: 0.9447, Train Acc: 66.63% | Val Loss: 0.8617, Val
Acc: 69.65%
  Epoch [09/20] | Train Loss: 0.9090, Train Acc: 67.81% | Val Loss: 0.8747, Val
Acc: 69.35%
  Epoch [10/20] | Train Loss: 0.8843, Train Acc: 68.94% | Val Loss: 0.8439, Val
Acc: 70.53%
  Epoch [11/20] | Train Loss: 0.8463, Train Acc: 70.31% | Val Loss: 0.8251, Val
Acc: 71.02%
  Epoch [12/20] | Train Loss: 0.8386, Train Acc: 70.63% | Val Loss: 0.7867, Val
Acc: 72.95%
  Epoch [13/20] | Train Loss: 0.8128, Train Acc: 71.79% | Val Loss: 0.7805, Val
Acc: 72.87%
  Epoch [14/20] | Train Loss: 0.7882, Train Acc: 72.33% | Val Loss: 0.7755, Val
Acc: 72.99%
  Epoch [15/20] | Train Loss: 0.7731, Train Acc: 72.87% | Val Loss: 0.7745, Val
Acc: 73.33%
  Epoch [16/20] | Train Loss: 0.7623, Train Acc: 73.36% | Val Loss: 0.7311, Val
Acc: 74.58%
  Epoch [17/20] | Train Loss: 0.7346, Train Acc: 74.39% | Val Loss: 0.7236, Val
Acc: 74.29%
  Epoch [18/20] | Train Loss: 0.7283, Train Acc: 74.49% | Val Loss: 0.7335, Val
Acc: 75.20%
  Epoch [19/20] | Train Loss: 0.7107, Train Acc: 74.90% | Val Loss: 0.7147, Val
Acc: 75.19%
  Epoch [20/20] | Train Loss: 0.6988, Train Acc: 75.74% | Val Loss: 0.7336, Val
Acc: 74.70%
FINISHED EXPERIMENT: SimpleCNN_SGD_Momentum
   Total Time: 471.53 seconds
STARTING EXPERIMENT: SimpleCNN_SGD_Nesterov
  Epoch [01/20] | Train Loss: 1.7453, Train Acc: 36.52% | Val Loss: 1.4171, Val
Acc: 47.83%
```

```
  Epoch [02/20] | Train Loss: 1.3826, Train Acc: 50.05% | Val Loss: 1.2235, Val
Acc: 56.64%
  Epoch [03/20] | Train Loss: 1.2133, Train Acc: 56.52% | Val Loss: 1.0973, Val
Acc: 61.44%
  Epoch [04/20] | Train Loss: 1.1065, Train Acc: 60.49% | Val Loss: 1.0091, Val
Acc: 64.98%
  Epoch [05/20] | Train Loss: 1.0324, Train Acc: 63.39% | Val Loss: 0.9355, Val
Acc: 67.23%
  Epoch [06/20] | Train Loss: 0.9768, Train Acc: 65.60% | Val Loss: 0.9281, Val
Acc: 68.11%
  Epoch [07/20] | Train Loss: 0.9265, Train Acc: 67.22% | Val Loss: 0.8336, Val
Acc: 71.27%
  Epoch [08/20] | Train Loss: 0.8948, Train Acc: 68.45% | Val Loss: 0.8600, Val
Acc: 70.69%
  Epoch [09/20] | Train Loss: 0.8641, Train Acc: 69.55% | Val Loss: 0.8185, Val
Acc: 71.88%
  Epoch [10/20] | Train Loss: 0.8373, Train Acc: 70.56% | Val Loss: 0.7865, Val
Acc: 72.50%
  Epoch [11/20] | Train Loss: 0.8128, Train Acc: 71.63% | Val Loss: 0.7798, Val
Acc: 73.07%
  Epoch [12/20] | Train Loss: 0.7931, Train Acc: 72.29% | Val Loss: 0.7639, Val
Acc: 73.39%
  Epoch [13/20] | Train Loss: 0.7651, Train Acc: 73.07% | Val Loss: 0.7617, Val
Acc: 73.53%
  Epoch [14/20] | Train Loss: 0.7583, Train Acc: 73.51% | Val Loss: 0.7345, Val
Acc: 74.63%
  Epoch [15/20] | Train Loss: 0.7416, Train Acc: 74.00% | Val Loss: 0.7200, Val
Acc: 74.69%
  Epoch [16/20] | Train Loss: 0.7245, Train Acc: 74.71% | Val Loss: 0.7163, Val
Acc: 75.51%
  Epoch [17/20] | Train Loss: 0.7095, Train Acc: 75.21% | Val Loss: 0.7017, Val
Acc: 75.59%
  Epoch [18/20] | Train Loss: 0.6984, Train Acc: 75.46% | Val Loss: 0.6816, Val
Acc: 76.10%
  Epoch [19/20] | Train Loss: 0.6883, Train Acc: 76.05% | Val Loss: 0.7082, Val
Acc: 75.59%
  Epoch [20/20] | Train Loss: 0.6770, Train Acc: 76.55% | Val Loss: 0.6720, Val
Acc: 76.86%
FINISHED EXPERIMENT: SimpleCNN_SGD_Nesterov
   Total Time: 496.31 seconds
STARTING EXPERIMENT: ResNet-18_Adam
  Epoch [01/20] | Train Loss: 1.0377, Train Acc: 64.38% | Val Loss: 1.0322, Val
Acc: 66.18%
  Epoch [02/20] | Train Loss: 0.7384, Train Acc: 74.85% | Val Loss: 0.6749, Val
Acc: 76.39%
  Epoch [03/20] | Train Loss: 0.6635, Train Acc: 77.45% | Val Loss: 0.6767, Val
Acc: 76.74%
  Epoch [04/20] | Train Loss: 0.5959, Train Acc: 79.75% | Val Loss: 0.6110, Val
```

```
Acc: 79.19%
  Epoch [05/20] | Train Loss: 0.5573, Train Acc: 80.88% | Val Loss: 0.5653, Val
Acc: 80.54%
  Epoch [06/20] | Train Loss: 0.5283, Train Acc: 81.90% | Val Loss: 0.5369, Val
Acc: 81.38%
  Epoch [07/20] | Train Loss: 0.4945, Train Acc: 82.97% | Val Loss: 0.5522, Val
Acc: 81.54%
  Epoch [08/20] | Train Loss: 0.4699, Train Acc: 83.91% | Val Loss: 0.5185, Val
Acc: 82.71%
  Epoch [09/20] | Train Loss: 0.4543, Train Acc: 84.46% | Val Loss: 0.4969, Val
Acc: 82.87%
  Epoch [10/20] | Train Loss: 0.4300, Train Acc: 85.06% | Val Loss: 0.5396, Val
Acc: 81.62%
  Epoch [11/20] | Train Loss: 0.4178, Train Acc: 85.52% | Val Loss: 0.5157, Val
Acc: 82.99%
  Epoch [12/20] | Train Loss: 0.3947, Train Acc: 86.38% | Val Loss: 0.4882, Val
Acc: 83.82%
  Epoch [13/20] | Train Loss: 0.3963, Train Acc: 86.39% | Val Loss: 0.5016, Val
Acc: 83.40%
  Epoch [14/20] | Train Loss: 0.3759, Train Acc: 87.00% | Val Loss: 0.4979, Val
Acc: 83.68%
  Epoch [15/20] | Train Loss: 0.3561, Train Acc: 87.65% | Val Loss: 0.4560, Val
Acc: 85.21%
  Epoch [16/20] | Train Loss: 0.3875, Train Acc: 86.77% | Val Loss: 0.4810, Val
Acc: 84.03%
  Epoch [17/20] | Train Loss: 0.3467, Train Acc: 87.98% | Val Loss: 0.4525, Val
Acc: 85.20%
  Epoch [18/20] | Train Loss: 0.3226, Train Acc: 88.69% | Val Loss: 0.4684, Val
Acc: 84.50%
  Epoch [19/20] | Train Loss: 0.3173, Train Acc: 88.98% | Val Loss: 0.4704, Val
Acc: 85.00%
  Epoch [20/20] | Train Loss: 0.3090, Train Acc: 89.27% | Val Loss: 0.4481, Val
Acc: 85.45%
FINISHED EXPERIMENT: ResNet-18_Adam
   Total Time: 524.84 seconds
STARTING EXPERIMENT: ResNet-18_SGD_Vanilla
  Epoch [01/20] | Train Loss: 1.2775, Train Acc: 54.84% | Val Loss: 0.9478, Val
Acc: 66.49%
  Epoch [02/20] | Train Loss: 0.8886, Train Acc: 68.95% | Val Loss: 0.7872, Val
Acc: 72.67%
  Epoch [03/20] | Train Loss: 0.7684, Train Acc: 73.19% | Val Loss: 0.7098, Val
Acc: 75.48%
  Epoch [04/20] | Train Loss: 0.6961, Train Acc: 75.69% | Val Loss: 0.6680, Val
Acc: 76.72%
  Epoch [05/20] | Train Loss: 0.6439, Train Acc: 77.35% | Val Loss: 0.6294, Val
Acc: 78.25%
  Epoch [06/20] | Train Loss: 0.6027, Train Acc: 78.87% | Val Loss: 0.6235, Val
Acc: 78.39%
```

```
  Epoch [07/20] | Train Loss: 0.5661, Train Acc: 80.13% | Val Loss: 0.5933, Val
Acc: 79.83%
  Epoch [08/20] | Train Loss: 0.5375, Train Acc: 81.13% | Val Loss: 0.5719, Val
Acc: 80.55%
  Epoch [09/20] | Train Loss: 0.5118, Train Acc: 82.07% | Val Loss: 0.5540, Val
Acc: 81.05%
  Epoch [10/20] | Train Loss: 0.4868, Train Acc: 82.91% | Val Loss: 0.5519, Val
Acc: 81.05%
  Epoch [11/20] | Train Loss: 0.4733, Train Acc: 83.36% | Val Loss: 0.5458, Val
Acc: 81.69%
  Epoch [12/20] | Train Loss: 0.4469, Train Acc: 84.26% | Val Loss: 0.5385, Val
Acc: 81.73%
  Epoch [13/20] | Train Loss: 0.4273, Train Acc: 84.86% | Val Loss: 0.5324, Val
Acc: 81.97%
  Epoch [14/20] | Train Loss: 0.4094, Train Acc: 85.29% | Val Loss: 0.5270, Val
Acc: 82.34%
  Epoch [15/20] | Train Loss: 0.3956, Train Acc: 85.96% | Val Loss: 0.5501, Val
Acc: 82.20%
  Epoch [16/20] | Train Loss: 0.3791, Train Acc: 86.48% | Val Loss: 0.5281, Val
Acc: 82.90%
  Epoch [17/20] | Train Loss: 0.3627, Train Acc: 86.96% | Val Loss: 0.5195, Val
Acc: 82.91%
  Epoch [18/20] | Train Loss: 0.3523, Train Acc: 87.46% | Val Loss: 0.5210, Val
Acc: 82.89%
  Epoch [19/20] | Train Loss: 0.3382, Train Acc: 87.89% | Val Loss: 0.5293, Val
Acc: 82.62%
  Epoch [20/20] | Train Loss: 0.3273, Train Acc: 88.26% | Val Loss: 0.5279, Val
Acc: 82.87%
FINISHED EXPERIMENT: ResNet-18_SGD_Vanilla
   Total Time: 477.26 seconds
STARTING EXPERIMENT: ResNet-18_SGD_Momentum
  Epoch [01/20] | Train Loss: 1.0377, Train Acc: 64.16% | Val Loss: 0.7368, Val
Acc: 74.72%
  Epoch [02/20] | Train Loss: 0.7057, Train Acc: 75.66% | Val Loss: 0.6512, Val
Acc: 77.44%
  Epoch [03/20] | Train Loss: 0.6135, Train Acc: 78.96% | Val Loss: 0.5900, Val
Acc: 79.69%
  Epoch [04/20] | Train Loss: 0.5584, Train Acc: 80.77% | Val Loss: 0.5718, Val
Acc: 80.92%
  Epoch [05/20] | Train Loss: 0.5082, Train Acc: 82.40% | Val Loss: 0.5827, Val
Acc: 80.20%
  Epoch [06/20] | Train Loss: 0.4819, Train Acc: 83.36% | Val Loss: 0.5086, Val
Acc: 82.70%
  Epoch [07/20] | Train Loss: 0.4470, Train Acc: 84.40% | Val Loss: 0.5236, Val
Acc: 82.61%
  Epoch [08/20] | Train Loss: 0.4270, Train Acc: 85.16% | Val Loss: 0.4888, Val
Acc: 83.38%
  Epoch [09/20] | Train Loss: 0.4111, Train Acc: 85.77% | Val Loss: 0.4946, Val
```

```
Acc: 83.69%
  Epoch [10/20] | Train Loss: 0.3855, Train Acc: 86.52% | Val Loss: 0.5069, Val
Acc: 82.88%
  Epoch [11/20] | Train Loss: 0.3676, Train Acc: 87.24% | Val Loss: 0.5076, Val
Acc: 83.66%
  Epoch [12/20] | Train Loss: 0.3514, Train Acc: 87.53% | Val Loss: 0.4915, Val
Acc: 84.28%
  Epoch [13/20] | Train Loss: 0.3475, Train Acc: 87.78% | Val Loss: 0.5013, Val
Acc: 83.80%
  Epoch [14/20] | Train Loss: 0.3264, Train Acc: 88.62% | Val Loss: 0.5239, Val
Acc: 83.27%
  Epoch [15/20] | Train Loss: 0.3171, Train Acc: 88.91% | Val Loss: 0.5194, Val
Acc: 83.59%
  Epoch [16/20] | Train Loss: 0.3081, Train Acc: 89.05% | Val Loss: 0.4679, Val
Acc: 84.98%
  Epoch [17/20] | Train Loss: 0.2937, Train Acc: 89.59% | Val Loss: 0.5295, Val
Acc: 83.71%
  Epoch [18/20] | Train Loss: 0.2826, Train Acc: 90.00% | Val Loss: 0.5033, Val
Acc: 84.40%
  Epoch [19/20] | Train Loss: 0.2758, Train Acc: 90.20% | Val Loss: 0.5162, Val
Acc: 84.38%
  Epoch [20/20] | Train Loss: 0.2621, Train Acc: 90.65% | Val Loss: 0.5051, Val
Acc: 84.32%
FINISHED EXPERIMENT: ResNet-18_SGD_Momentum
   Total Time: 541.16 seconds
STARTING EXPERIMENT: ResNet-18_SGD_Nesterov
  Epoch [01/20] | Train Loss: 0.9954, Train Acc: 65.67% | Val Loss: 0.7445, Val
Acc: 74.48%
  Epoch [02/20] | Train Loss: 0.6953, Train Acc: 76.15% | Val Loss: 0.6431, Val
Acc: 78.87%
  Epoch [03/20] | Train Loss: 0.6069, Train Acc: 79.29% | Val Loss: 0.6136, Val
Acc: 79.65%
  Epoch [04/20] | Train Loss: 0.5547, Train Acc: 80.86% | Val Loss: 0.5502, Val
Acc: 81.12%
  Epoch [05/20] | Train Loss: 0.5073, Train Acc: 82.44% | Val Loss: 0.5398, Val
Acc: 81.19%
  Epoch [06/20] | Train Loss: 0.4729, Train Acc: 83.56% | Val Loss: 0.5171, Val
Acc: 82.56%
  Epoch [07/20] | Train Loss: 0.4482, Train Acc: 84.21% | Val Loss: 0.4750, Val
Acc: 83.69%
  Epoch [08/20] | Train Loss: 0.4230, Train Acc: 85.17% | Val Loss: 0.5136, Val
Acc: 83.08%
  Epoch [09/20] | Train Loss: 0.4059, Train Acc: 85.69% | Val Loss: 0.5154, Val
Acc: 82.44%
  Epoch [10/20] | Train Loss: 0.3832, Train Acc: 86.36% | Val Loss: 0.4751, Val
Acc: 84.28%
  Epoch [11/20] | Train Loss: 0.3688, Train Acc: 87.00% | Val Loss: 0.4738, Val
Acc: 84.35%
```

```
  Epoch [12/20] | Train Loss: 0.3508, Train Acc: 87.52% | Val Loss: 0.5006, Val
Acc: 84.00%
  Epoch [13/20] | Train Loss: 0.3352, Train Acc: 88.21% | Val Loss: 0.4858, Val
Acc: 84.36%
  Epoch [14/20] | Train Loss: 0.3239, Train Acc: 88.69% | Val Loss: 0.4902, Val
Acc: 84.09%
  Epoch [15/20] | Train Loss: 0.3088, Train Acc: 89.20% | Val Loss: 0.4657, Val
Acc: 84.85%
  Epoch [16/20] | Train Loss: 0.3075, Train Acc: 89.25% | Val Loss: 0.4778, Val
Acc: 84.48%
  Epoch [17/20] | Train Loss: 0.2877, Train Acc: 89.96% | Val Loss: 0.4795, Val
Acc: 84.74%
  Epoch [18/20] | Train Loss: 0.2779, Train Acc: 90.24% | Val Loss: 0.4805, Val
Acc: 85.02%
  Epoch [19/20] | Train Loss: 0.2691, Train Acc: 90.57% | Val Loss: 0.4795, Val
Acc: 85.00%
  Epoch [20/20] | Train Loss: 0.2610, Train Acc: 90.65% | Val Loss: 0.4923, Val
Acc: 84.63%
FINISHED EXPERIMENT: ResNet-18_SGD_Nesterov
   Total Time: 616.12 seconds
All 8 experiments completed!
```

[10]:

```python
# Dictionary to store all results
experiment_results_with_schedulers = {}

# Select specific configurations to test with schedulers
# To keep experiments manageable, we'll test schedulers with Adam optimizer only
model_configs_for_scheduler = [
    {'name': 'SimpleCNN', 'builder': SimpleCNN},
    {'name': 'ResNet-18', 'builder': get_resnet_model}
]

optimizer_config_for_scheduler = {
    'name': 'Adam',
    'builder': lambda params: optim.Adam(params, lr=LR_ADAM)
}

if train_loader is None or test_loader is None:
    print("FATAL: Data loaders were not initialized. Stopping experiment.")
else:
    print("STARTING EXPERIMENTS WITH LEARNING RATE SCHEDULERS")

    for model_config in model_configs_for_scheduler:
        for scheduler_config in scheduler_configs:
            experiment_name =␣
 ↪f"{model_config['name']}_{optimizer_config_for_scheduler['name']}_{scheduler_config['name']
            print(f"\nSTARTING EXPERIMENT: {experiment_name}")
```

```
            print("-" * 60)

            # Initialize model, optimizer, and scheduler
            model = model_config['builder']().to(DEVICE)
            optimizer = optimizer_config_for_scheduler['builder'](model.
↪parameters())
            scheduler = scheduler_config['builder'](optimizer)

            # Store history for this run
            history = {
                'train_loss': [], 'train_acc': [],
                'val_loss': [], 'val_acc': [],
                'learning_rates': []
            }

            start_time = time.time()

            # Run training epochs
            for epoch in range(NUM_EPOCHS):
                # Train
                train_loss, train_acc = train_one_epoch_with_scheduler(
                    model, train_loader, optimizer, criterion, DEVICE)

                # Validate
                val_loss, val_acc, _, _ = evaluate_model(
                    model, test_loader, criterion, DEVICE)

                # Log results
                history['train_loss'].append(train_loss)
                history['train_acc'].append(train_acc)
                history['val_loss'].append(val_loss)
                history['val_acc'].append(val_acc)

                # Get current learning rate
                current_lr = optimizer.param_groups[0]['lr']
                history['learning_rates'].append(current_lr)

                # Step the scheduler
                if scheduler is not None:
                    if isinstance(scheduler, optim.lr_scheduler.
↪ReduceLROnPlateau):
                        scheduler.step(val_loss)
                    else:
                        scheduler.step()

                print(f"  Epoch [{epoch+1:02d}/{NUM_EPOCHS}] | "
```

```
                      f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.
↪2f}% | "
                      f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}% | "
                      f"LR: {current_lr:.6f}")

            end_time = time.time()
            print(f"\nFINISHED EXPERIMENT: {experiment_name}")
            print(f"  Total Time: {end_time - start_time:.2f} seconds")
            print(f"  Best Val Acc: {max(history['val_acc']):.2f}%")

            # Store results
            experiment_results_with_schedulers[experiment_name] = {
                'model_state': deepcopy(model.state_dict()),
                'model_config': model_config,
                'history': history,
                'time': end_time - start_time,
                'scheduler_name': scheduler_config['name']
            }

            # Clean up memory
            del model
            del optimizer
            del scheduler
            if DEVICE == 'cuda':
                torch.cuda.empty_cache()

    print("\n" + "="*80)
    print("All Schedular Experiments Done.")
    print("="*80)

# Enhanced Visualization Function for Schedulers

def plot_history_with_lr(history, title):
    """Plot training history including learning rate"""
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24, 6))

    # Accuracy
    ax1.plot(history['train_acc'], label='Train Accuracy', linewidth=2)
    ax1.plot(history['val_acc'], label='Validation Accuracy', linewidth=2)
    ax1.set_title(f'Accuracy\n{title}', fontsize=12, fontweight='bold')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy (%)')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Loss
    ax2.plot(history['train_loss'], label='Train Loss', linewidth=2)
```

```
    ax2.plot(history['val_loss'], label='Validation Loss', linewidth=2)
    ax2.set_title(f'Loss\n{title}', fontsize=12, fontweight='bold')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    # Learning Rate
    ax3.plot(history['learning_rates'], color='red', linewidth=2, marker='o',␣
 ↪markersize=4)
    ax3.set_title(f'Learning Rate Schedule\n{title}', fontsize=12,␣
 ↪fontweight='bold')
    ax3.set_xlabel('Epoch')
    ax3.set_ylabel('Learning Rate')
    ax3.set_yscale('log')  # Log scale to better visualize LR changes
    ax3.grid(True, alpha=0.3)

    plt.suptitle(f'Training History with LR Schedule: {title}', fontsize=16,␣
 ↪fontweight='bold')
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

# Analyze and Visualize Scheduler Results
if experiment_results_with_schedulers:
    print("\n" + "="*80)
    print("Analyzing Schedular Experiments")
    print("="*80 + "\n")

    for experiment_name, results in experiment_results_with_schedulers.items():
        print(f"\nAnalyzing: {experiment_name}")
        print("-" * 60)

        # Plot enhanced training history with LR
        print(f"[Plotting] Training History with LR for {experiment_name}...")
        plot_history_with_lr(results['history'], title=experiment_name)

        # Print summary statistics
        best_val_acc = max(results['history']['val_acc'])
        best_epoch = results['history']['val_acc'].index(best_val_acc) + 1
        final_val_acc = results['history']['val_acc'][-1]

        print(f"\n  Summary Statistics:")
        print(f"- Best Validation Accuracy: {best_val_acc:.2f}% (Epoch␣
 ↪{best_epoch})")
        print(f"- Final Validation Accuracy: {final_val_acc:.2f}%")
        print(f"- Training Time: {results['time']:.2f} seconds")
        print(f"- Scheduler Used: {results['scheduler_name']}")
```

```python
# Comparison Table for Schedulers
def create_scheduler_comparison_table(results_dict):
    comparison_data = []
    for exp_name, results in results_dict.items():
        comparison_data.append({
            'Experiment': exp_name,
            'Scheduler': results['scheduler_name'],
            'Best Val Acc (%)': f"{max(results['history']['val_acc']):.2f}",
            'Final Val Acc (%)': f"{results['history']['val_acc'][-1]:.2f}",
            'Time (s)': f"{results['time']:.2f}",
            'Final LR': f"{results['history']['learning_rates'][-1]:.6f}"
        })

    df = pd.DataFrame(comparison_data)
    print("Schedular Comparison Table")
    print(df.to_string(index=False))

    return df

if experiment_results_with_schedulers:
    scheduler_comparison_df =␣
 ↪create_scheduler_comparison_table(experiment_results_with_schedulers)
```

```
STARTING EXPERIMENTS WITH LEARNING RATE SCHEDULERS

STARTING EXPERIMENT: SimpleCNN_Adam_None
------------------------------------------------------------
  Epoch [01/20] | Train Loss: 1.6152, Train Acc: 41.25% | Val Loss: 1.3441, Val
Acc: 52.00% | LR: 0.001000
  Epoch [02/20] | Train Loss: 1.3233, Train Acc: 52.00% | Val Loss: 1.1465, Val
Acc: 59.03% | LR: 0.001000
  Epoch [03/20] | Train Loss: 1.1873, Train Acc: 57.67% | Val Loss: 1.0365, Val
Acc: 62.94% | LR: 0.001000
  Epoch [04/20] | Train Loss: 1.0995, Train Acc: 61.08% | Val Loss: 0.9747, Val
Acc: 65.81% | LR: 0.001000
  Epoch [05/20] | Train Loss: 1.0279, Train Acc: 63.60% | Val Loss: 0.9220, Val
Acc: 67.37% | LR: 0.001000
  Epoch [06/20] | Train Loss: 0.9813, Train Acc: 65.21% | Val Loss: 0.8892, Val
Acc: 69.00% | LR: 0.001000
  Epoch [07/20] | Train Loss: 0.9444, Train Acc: 66.53% | Val Loss: 0.8582, Val
Acc: 69.93% | LR: 0.001000
  Epoch [08/20] | Train Loss: 0.9105, Train Acc: 67.72% | Val Loss: 0.8416, Val
Acc: 70.85% | LR: 0.001000
  Epoch [09/20] | Train Loss: 0.8896, Train Acc: 68.69% | Val Loss: 0.8255, Val
Acc: 71.72% | LR: 0.001000
  Epoch [10/20] | Train Loss: 0.8640, Train Acc: 69.57% | Val Loss: 0.8143, Val
Acc: 72.39% | LR: 0.001000
```

```
  Epoch [11/20] | Train Loss: 0.8424, Train Acc: 70.65% | Val Loss: 0.7923, Val
Acc: 72.78% | LR: 0.001000
  Epoch [12/20] | Train Loss: 0.8260, Train Acc: 71.04% | Val Loss: 0.8292, Val
Acc: 71.71% | LR: 0.001000
  Epoch [13/20] | Train Loss: 0.8081, Train Acc: 71.49% | Val Loss: 0.7778, Val
Acc: 73.24% | LR: 0.001000
  Epoch [14/20] | Train Loss: 0.7913, Train Acc: 72.22% | Val Loss: 0.7554, Val
Acc: 73.83% | LR: 0.001000
  Epoch [15/20] | Train Loss: 0.7776, Train Acc: 72.72% | Val Loss: 0.7771, Val
Acc: 73.61% | LR: 0.001000
  Epoch [16/20] | Train Loss: 0.7673, Train Acc: 73.09% | Val Loss: 0.7388, Val
Acc: 74.84% | LR: 0.001000
  Epoch [17/20] | Train Loss: 0.7544, Train Acc: 73.43% | Val Loss: 0.7613, Val
Acc: 73.54% | LR: 0.001000
  Epoch [18/20] | Train Loss: 0.7441, Train Acc: 73.91% | Val Loss: 0.7234, Val
Acc: 75.14% | LR: 0.001000
  Epoch [19/20] | Train Loss: 0.7302, Train Acc: 74.40% | Val Loss: 0.7347, Val
Acc: 74.99% | LR: 0.001000
  Epoch [20/20] | Train Loss: 0.7227, Train Acc: 74.69% | Val Loss: 0.7216, Val
Acc: 75.09% | LR: 0.001000

FINISHED EXPERIMENT: SimpleCNN_Adam_None
  Total Time: 604.44 seconds
  Best Val Acc: 75.14%

STARTING EXPERIMENT: SimpleCNN_Adam_StepLR
------------------------------------------------------------
  Epoch [01/20] | Train Loss: 1.6321, Train Acc: 40.49% | Val Loss: 1.3837, Val
Acc: 50.44% | LR: 0.001000
  Epoch [02/20] | Train Loss: 1.3428, Train Acc: 52.01% | Val Loss: 1.1750, Val
Acc: 57.25% | LR: 0.001000
  Epoch [03/20] | Train Loss: 1.2109, Train Acc: 56.84% | Val Loss: 1.0797, Val
Acc: 61.42% | LR: 0.001000
  Epoch [04/20] | Train Loss: 1.1195, Train Acc: 60.33% | Val Loss: 1.0015, Val
Acc: 64.36% | LR: 0.001000
  Epoch [05/20] | Train Loss: 1.0502, Train Acc: 62.76% | Val Loss: 0.9510, Val
Acc: 65.94% | LR: 0.001000
  Epoch [06/20] | Train Loss: 1.0059, Train Acc: 64.24% | Val Loss: 0.9121, Val
Acc: 67.68% | LR: 0.001000
  Epoch [07/20] | Train Loss: 0.9619, Train Acc: 65.91% | Val Loss: 0.8837, Val
Acc: 69.31% | LR: 0.001000
  Epoch [08/20] | Train Loss: 0.8834, Train Acc: 68.87% | Val Loss: 0.8424, Val
Acc: 70.86% | LR: 0.000100
  Epoch [09/20] | Train Loss: 0.8683, Train Acc: 69.76% | Val Loss: 0.8338, Val
Acc: 70.91% | LR: 0.000100
  Epoch [10/20] | Train Loss: 0.8637, Train Acc: 69.55% | Val Loss: 0.8286, Val
Acc: 71.02% | LR: 0.000100
  Epoch [11/20] | Train Loss: 0.8564, Train Acc: 69.77% | Val Loss: 0.8242, Val
```

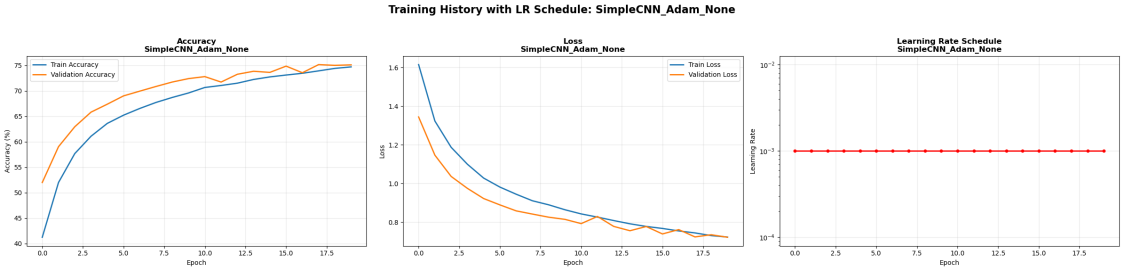
```
Acc: 71.42% | LR: 0.000100
  Epoch [12/20] | Train Loss: 0.8549, Train Acc: 69.92% | Val Loss: 0.8201, Val
Acc: 71.54% | LR: 0.000100
  Epoch [13/20] | Train Loss: 0.8463, Train Acc: 70.20% | Val Loss: 0.8168, Val
Acc: 71.69% | LR: 0.000100
  Epoch [14/20] | Train Loss: 0.8404, Train Acc: 70.34% | Val Loss: 0.8083, Val
Acc: 72.05% | LR: 0.000100
  Epoch [15/20] | Train Loss: 0.8339, Train Acc: 70.65% | Val Loss: 0.8073, Val
Acc: 71.92% | LR: 0.000010
  Epoch [16/20] | Train Loss: 0.8290, Train Acc: 70.84% | Val Loss: 0.8075, Val
Acc: 71.90% | LR: 0.000010
  Epoch [17/20] | Train Loss: 0.8298, Train Acc: 70.65% | Val Loss: 0.8061, Val
Acc: 71.91% | LR: 0.000010
  Epoch [18/20] | Train Loss: 0.8304, Train Acc: 70.59% | Val Loss: 0.8062, Val
Acc: 71.99% | LR: 0.000010
  Epoch [19/20] | Train Loss: 0.8268, Train Acc: 70.91% | Val Loss: 0.8059, Val
Acc: 71.92% | LR: 0.000010
  Epoch [20/20] | Train Loss: 0.8253, Train Acc: 70.95% | Val Loss: 0.8052, Val
Acc: 72.01% | LR: 0.000010

FINISHED EXPERIMENT: SimpleCNN_Adam_StepLR
  Total Time: 587.59 seconds
  Best Val Acc: 72.05%

STARTING EXPERIMENT: SimpleCNN_Adam_ReduceLROnPlateau
------------------------------------------------------------

c:\Users\mfaha\miniconda3\envs\cuda_env\lib\site-
packages\torch\optim\lr_scheduler.py:62: UserWarning: The verbose parameter is
deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(
  Epoch [01/20] | Train Loss: 1.6145, Train Acc: 41.33% | Val Loss: 1.2910, Val
Acc: 53.36% | LR: 0.001000
  Epoch [02/20] | Train Loss: 1.3124, Train Acc: 53.08% | Val Loss: 1.1207, Val
Acc: 59.88% | LR: 0.001000
  Epoch [03/20] | Train Loss: 1.1845, Train Acc: 57.96% | Val Loss: 1.0682, Val
Acc: 61.95% | LR: 0.001000
  Epoch [04/20] | Train Loss: 1.0998, Train Acc: 60.93% | Val Loss: 0.9839, Val
Acc: 65.21% | LR: 0.001000
  Epoch [05/20] | Train Loss: 1.0357, Train Acc: 63.53% | Val Loss: 0.9474, Val
Acc: 66.95% | LR: 0.001000
  Epoch [06/20] | Train Loss: 0.9875, Train Acc: 65.12% | Val Loss: 0.9167, Val
Acc: 67.57% | LR: 0.001000
  Epoch [07/20] | Train Loss: 0.9578, Train Acc: 66.04% | Val Loss: 0.8764, Val
Acc: 69.16% | LR: 0.001000
  Epoch [08/20] | Train Loss: 0.9216, Train Acc: 67.49% | Val Loss: 0.8875, Val
Acc: 68.92% | LR: 0.001000
  Epoch [09/20] | Train Loss: 0.8965, Train Acc: 68.62% | Val Loss: 0.8398, Val
```

```
Acc: 70.72% | LR: 0.001000
  Epoch [10/20] | Train Loss: 0.8699, Train Acc: 69.35% | Val Loss: 0.8110, Val
Acc: 71.77% | LR: 0.001000
  Epoch [11/20] | Train Loss: 0.8496, Train Acc: 70.14% | Val Loss: 0.7992, Val
Acc: 71.63% | LR: 0.001000
  Epoch [12/20] | Train Loss: 0.8251, Train Acc: 70.93% | Val Loss: 0.7947, Val
Acc: 72.65% | LR: 0.001000
  Epoch [13/20] | Train Loss: 0.8180, Train Acc: 71.41% | Val Loss: 0.7899, Val
Acc: 72.83% | LR: 0.001000
  Epoch [14/20] | Train Loss: 0.7993, Train Acc: 72.14% | Val Loss: 0.7652, Val
Acc: 73.40% | LR: 0.001000
  Epoch [15/20] | Train Loss: 0.7817, Train Acc: 72.70% | Val Loss: 0.7670, Val
Acc: 73.73% | LR: 0.001000
  Epoch [16/20] | Train Loss: 0.7724, Train Acc: 72.84% | Val Loss: 0.7620, Val
Acc: 73.67% | LR: 0.001000
  Epoch [17/20] | Train Loss: 0.7611, Train Acc: 73.48% | Val Loss: 0.7534, Val
Acc: 73.56% | LR: 0.001000
  Epoch [18/20] | Train Loss: 0.7488, Train Acc: 73.81% | Val Loss: 0.7331, Val
Acc: 74.61% | LR: 0.001000
  Epoch [19/20] | Train Loss: 0.7363, Train Acc: 74.22% | Val Loss: 0.7238, Val
Acc: 75.21% | LR: 0.001000
  Epoch [20/20] | Train Loss: 0.7251, Train Acc: 74.47% | Val Loss: 0.7201, Val
Acc: 75.15% | LR: 0.001000

FINISHED EXPERIMENT: SimpleCNN_Adam_ReduceLROnPlateau
  Total Time: 598.43 seconds
  Best Val Acc: 75.21%

STARTING EXPERIMENT: ResNet-18_Adam_None
------------------------------------------------------------
  Epoch [01/20] | Train Loss: 1.0253, Train Acc: 64.79% | Val Loss: 0.8074, Val
Acc: 72.37% | LR: 0.001000
  Epoch [02/20] | Train Loss: 0.7399, Train Acc: 74.90% | Val Loss: 0.6747, Val
Acc: 76.86% | LR: 0.001000
  Epoch [03/20] | Train Loss: 0.6575, Train Acc: 77.69% | Val Loss: 0.6334, Val
Acc: 77.72% | LR: 0.001000
  Epoch [04/20] | Train Loss: 0.5928, Train Acc: 79.76% | Val Loss: 0.5813, Val
Acc: 80.20% | LR: 0.001000
  Epoch [05/20] | Train Loss: 0.5534, Train Acc: 81.07% | Val Loss: 0.5905, Val
Acc: 80.03% | LR: 0.001000
  Epoch [06/20] | Train Loss: 0.5204, Train Acc: 82.34% | Val Loss: 0.5550, Val
Acc: 80.95% | LR: 0.001000
  Epoch [07/20] | Train Loss: 0.4947, Train Acc: 83.13% | Val Loss: 0.5270, Val
Acc: 82.15% | LR: 0.001000
  Epoch [08/20] | Train Loss: 0.4941, Train Acc: 83.13% | Val Loss: 0.5538, Val
Acc: 81.22% | LR: 0.001000
  Epoch [09/20] | Train Loss: 0.4697, Train Acc: 84.11% | Val Loss: 0.5524, Val
Acc: 81.22% | LR: 0.001000
```

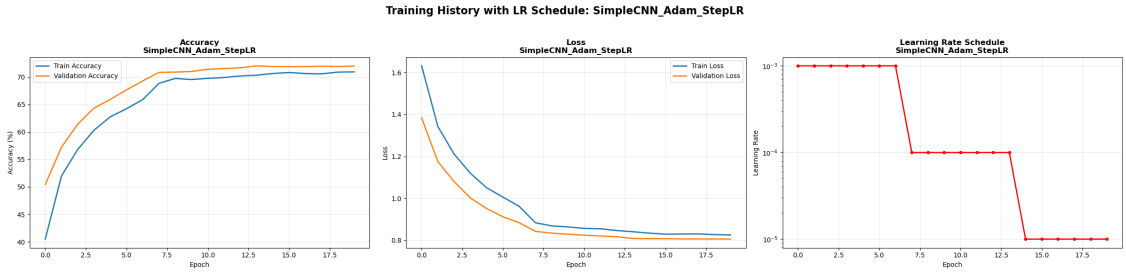
```
  Epoch [10/20] | Train Loss: 0.4341, Train Acc: 85.00% | Val Loss: 0.4779, Val
Acc: 84.06% | LR: 0.001000
  Epoch [11/20] | Train Loss: 0.4125, Train Acc: 85.79% | Val Loss: 0.4882, Val
Acc: 83.61% | LR: 0.001000
  Epoch [12/20] | Train Loss: 0.3952, Train Acc: 86.34% | Val Loss: 0.5135, Val
Acc: 83.38% | LR: 0.001000
  Epoch [13/20] | Train Loss: 0.3878, Train Acc: 86.60% | Val Loss: 0.4870, Val
Acc: 83.84% | LR: 0.001000
  Epoch [14/20] | Train Loss: 0.3711, Train Acc: 87.20% | Val Loss: 0.4617, Val
Acc: 84.63% | LR: 0.001000
  Epoch [15/20] | Train Loss: 0.3649, Train Acc: 87.32% | Val Loss: 0.4650, Val
Acc: 84.66% | LR: 0.001000
  Epoch [16/20] | Train Loss: 0.3476, Train Acc: 87.91% | Val Loss: 0.4852, Val
Acc: 84.28% | LR: 0.001000
  Epoch [17/20] | Train Loss: 0.3399, Train Acc: 88.10% | Val Loss: 0.4725, Val
Acc: 84.62% | LR: 0.001000
  Epoch [18/20] | Train Loss: 0.3301, Train Acc: 88.47% | Val Loss: 0.4619, Val
Acc: 85.10% | LR: 0.001000
  Epoch [19/20] | Train Loss: 0.3200, Train Acc: 88.84% | Val Loss: 0.4562, Val
Acc: 84.89% | LR: 0.001000
  Epoch [20/20] | Train Loss: 0.3013, Train Acc: 89.36% | Val Loss: 0.4785, Val
Acc: 84.48% | LR: 0.001000

FINISHED EXPERIMENT: ResNet-18_Adam_None
  Total Time: 642.29 seconds
  Best Val Acc: 85.10%

STARTING EXPERIMENT: ResNet-18_Adam_StepLR
------------------------------------------------------------
  Epoch [01/20] | Train Loss: 1.0226, Train Acc: 64.82% | Val Loss: 0.7784, Val
Acc: 73.93% | LR: 0.001000
  Epoch [02/20] | Train Loss: 0.7332, Train Acc: 75.05% | Val Loss: 0.6630, Val
Acc: 77.43% | LR: 0.001000
  Epoch [03/20] | Train Loss: 0.6480, Train Acc: 77.88% | Val Loss: 0.6788, Val
Acc: 77.57% | LR: 0.001000
  Epoch [04/20] | Train Loss: 0.5980, Train Acc: 79.66% | Val Loss: 0.6879, Val
Acc: 77.49% | LR: 0.001000
  Epoch [05/20] | Train Loss: 0.5489, Train Acc: 81.02% | Val Loss: 0.5980, Val
Acc: 79.98% | LR: 0.001000
  Epoch [06/20] | Train Loss: 0.5145, Train Acc: 82.41% | Val Loss: 0.5238, Val
Acc: 82.03% | LR: 0.001000
  Epoch [07/20] | Train Loss: 0.4918, Train Acc: 83.11% | Val Loss: 0.5237, Val
Acc: 82.25% | LR: 0.001000
  Epoch [08/20] | Train Loss: 0.3856, Train Acc: 86.64% | Val Loss: 0.4201, Val
Acc: 85.95% | LR: 0.000100
  Epoch [09/20] | Train Loss: 0.3468, Train Acc: 88.02% | Val Loss: 0.4133, Val
Acc: 86.24% | LR: 0.000100
  Epoch [10/20] | Train Loss: 0.3301, Train Acc: 88.52% | Val Loss: 0.4113, Val
```

```
Acc: 86.43% | LR: 0.000100
  Epoch [11/20] | Train Loss: 0.3195, Train Acc: 88.94% | Val Loss: 0.4049, Val
Acc: 86.65% | LR: 0.000100
  Epoch [12/20] | Train Loss: 0.3049, Train Acc: 89.56% | Val Loss: 0.4026, Val
Acc: 86.91% | LR: 0.000100
  Epoch [13/20] | Train Loss: 0.2942, Train Acc: 89.87% | Val Loss: 0.3998, Val
Acc: 86.71% | LR: 0.000100
  Epoch [14/20] | Train Loss: 0.2852, Train Acc: 89.93% | Val Loss: 0.4078, Val
Acc: 86.57% | LR: 0.000100
  Epoch [15/20] | Train Loss: 0.2722, Train Acc: 90.51% | Val Loss: 0.3989, Val
Acc: 86.86% | LR: 0.000010
  Epoch [16/20] | Train Loss: 0.2635, Train Acc: 90.82% | Val Loss: 0.3968, Val
Acc: 87.10% | LR: 0.000010
  Epoch [17/20] | Train Loss: 0.2625, Train Acc: 90.74% | Val Loss: 0.3963, Val
Acc: 87.12% | LR: 0.000010
  Epoch [18/20] | Train Loss: 0.2561, Train Acc: 91.10% | Val Loss: 0.4052, Val
Acc: 87.03% | LR: 0.000010
  Epoch [19/20] | Train Loss: 0.2582, Train Acc: 91.01% | Val Loss: 0.3986, Val
Acc: 87.13% | LR: 0.000010
  Epoch [20/20] | Train Loss: 0.2594, Train Acc: 90.96% | Val Loss: 0.4003, Val
Acc: 87.03% | LR: 0.000010

FINISHED EXPERIMENT: ResNet-18_Adam_StepLR
  Total Time: 768.41 seconds
  Best Val Acc: 87.13%

STARTING EXPERIMENT: ResNet-18_Adam_ReduceLROnPlateau
------------------------------------------------------------
  Epoch [01/20] | Train Loss: 1.0254, Train Acc: 64.92% | Val Loss: 0.7988, Val
Acc: 72.72% | LR: 0.001000
  Epoch [02/20] | Train Loss: 0.7386, Train Acc: 75.01% | Val Loss: 0.7239, Val
Acc: 75.50% | LR: 0.001000
  Epoch [03/20] | Train Loss: 0.6546, Train Acc: 77.59% | Val Loss: 0.6299, Val
Acc: 78.16% | LR: 0.001000
  Epoch [04/20] | Train Loss: 0.5959, Train Acc: 79.66% | Val Loss: 0.5774, Val
Acc: 80.24% | LR: 0.001000
  Epoch [05/20] | Train Loss: 0.5469, Train Acc: 81.27% | Val Loss: 0.5905, Val
Acc: 79.84% | LR: 0.001000
  Epoch [06/20] | Train Loss: 0.5139, Train Acc: 82.31% | Val Loss: 0.5194, Val
Acc: 82.52% | LR: 0.001000
  Epoch [07/20] | Train Loss: 0.5058, Train Acc: 82.73% | Val Loss: 0.5539, Val
Acc: 81.87% | LR: 0.001000
  Epoch [08/20] | Train Loss: 0.4632, Train Acc: 84.19% | Val Loss: 0.5010, Val
Acc: 83.08% | LR: 0.001000
  Epoch [09/20] | Train Loss: 0.4510, Train Acc: 84.46% | Val Loss: 0.4857, Val
Acc: 83.62% | LR: 0.001000
  Epoch [10/20] | Train Loss: 0.4350, Train Acc: 85.05% | Val Loss: 0.5655, Val
Acc: 82.11% | LR: 0.001000
```

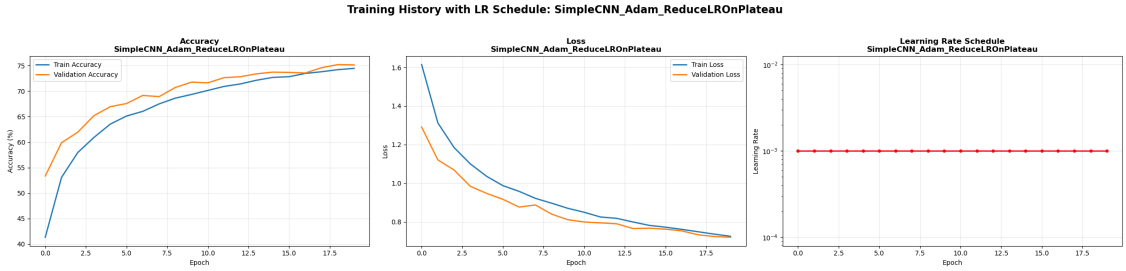
```
  Epoch [11/20] | Train Loss: 0.4117, Train Acc: 85.79% | Val Loss: 0.5224, Val
Acc: 82.62% | LR: 0.001000
  Epoch [12/20] | Train Loss: 0.3978, Train Acc: 86.30% | Val Loss: 0.4811, Val
Acc: 83.80% | LR: 0.001000
  Epoch [13/20] | Train Loss: 0.3793, Train Acc: 86.71% | Val Loss: 0.4864, Val
Acc: 83.82% | LR: 0.001000
  Epoch [14/20] | Train Loss: 0.3707, Train Acc: 87.17% | Val Loss: 0.4718, Val
Acc: 84.77% | LR: 0.001000
  Epoch [15/20] | Train Loss: 0.3579, Train Acc: 87.49% | Val Loss: 0.4839, Val
Acc: 83.91% | LR: 0.001000
  Epoch [16/20] | Train Loss: 0.3355, Train Acc: 88.41% | Val Loss: 0.4584, Val
Acc: 84.89% | LR: 0.001000
  Epoch [17/20] | Train Loss: 0.3340, Train Acc: 88.39% | Val Loss: 0.5605, Val
Acc: 82.75% | LR: 0.001000
  Epoch [18/20] | Train Loss: 0.3973, Train Acc: 86.34% | Val Loss: 0.4572, Val
Acc: 85.10% | LR: 0.001000
  Epoch [19/20] | Train Loss: 0.3191, Train Acc: 89.01% | Val Loss: 0.4286, Val
Acc: 86.14% | LR: 0.001000
  Epoch [20/20] | Train Loss: 0.3189, Train Acc: 88.83% | Val Loss: 0.4764, Val
Acc: 84.29% | LR: 0.001000

FINISHED EXPERIMENT: ResNet-18_Adam_ReduceLROnPlateau
  Total Time: 733.79 seconds
  Best Val Acc: 86.14%

================================================================================
All Schedular Experiments Done.
================================================================================

================================================================================
Analyzing Schedular Experiments
================================================================================


Analyzing: SimpleCNN_Adam_None
----------------------------------------------------------------
[Plotting] Training History with LR for SimpleCNN_Adam_None…
```



```
  Summary Statistics:
- Best Validation Accuracy: 75.14% (Epoch 18)
- Final Validation Accuracy: 75.09%
- Training Time: 604.44 seconds
- Scheduler Used: None

Analyzing: SimpleCNN_Adam_StepLR
----------------------------------------------------------------
[Plotting] Training History with LR for SimpleCNN_Adam_StepLR...
```




```
  Summary Statistics:
- Best Validation Accuracy: 72.05% (Epoch 14)
- Final Validation Accuracy: 72.01%
- Training Time: 587.59 seconds
- Scheduler Used: StepLR

Analyzing: SimpleCNN_Adam_ReduceLROnPlateau
----------------------------------------------------------------
[Plotting] Training History with LR for SimpleCNN_Adam_ReduceLROnPlateau...
```

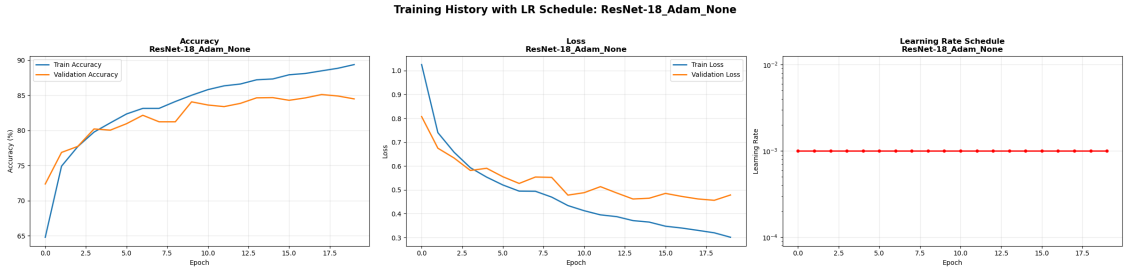



```
  Summary Statistics:
- Best Validation Accuracy: 75.21% (Epoch 19)
- Final Validation Accuracy: 75.15%
- Training Time: 598.43 seconds
```

- Scheduler Used: ReduceLROnPlateau

Analyzing: ResNet-18_Adam_None

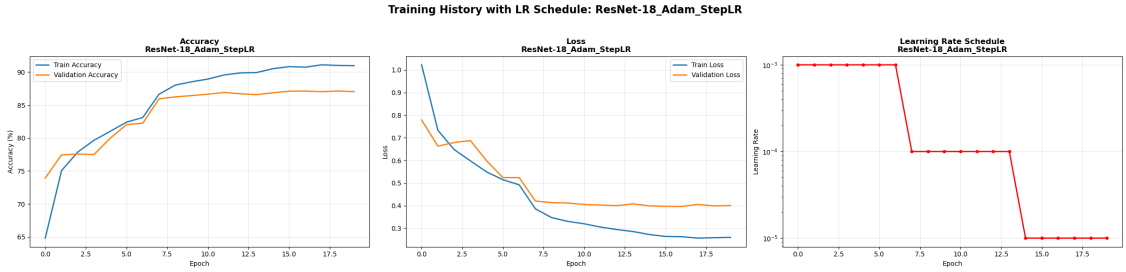
[Plotting] Training History with LR for ResNet-18_Adam_None...



  Summary Statistics:
- Best Validation Accuracy: 85.10% (Epoch 18)
- Final Validation Accuracy: 84.48%
- Training Time: 642.29 seconds
- Scheduler Used: None

Analyzing: ResNet-18_Adam_StepLR

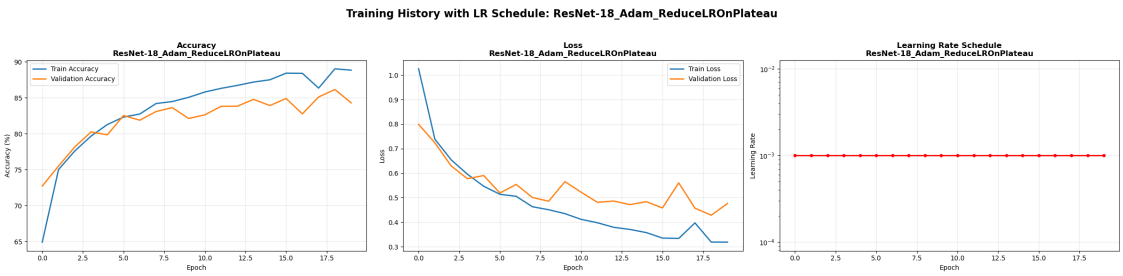
[Plotting] Training History with LR for ResNet-18_Adam_StepLR...



  Summary Statistics:
- Best Validation Accuracy: 87.13% (Epoch 19)
- Final Validation Accuracy: 87.03%
- Training Time: 768.41 seconds
- Scheduler Used: StepLR

Analyzing: ResNet-18_Adam_ReduceLROnPlateau

[Plotting] Training History with LR for ResNet-18_Adam_ReduceLROnPlateau...



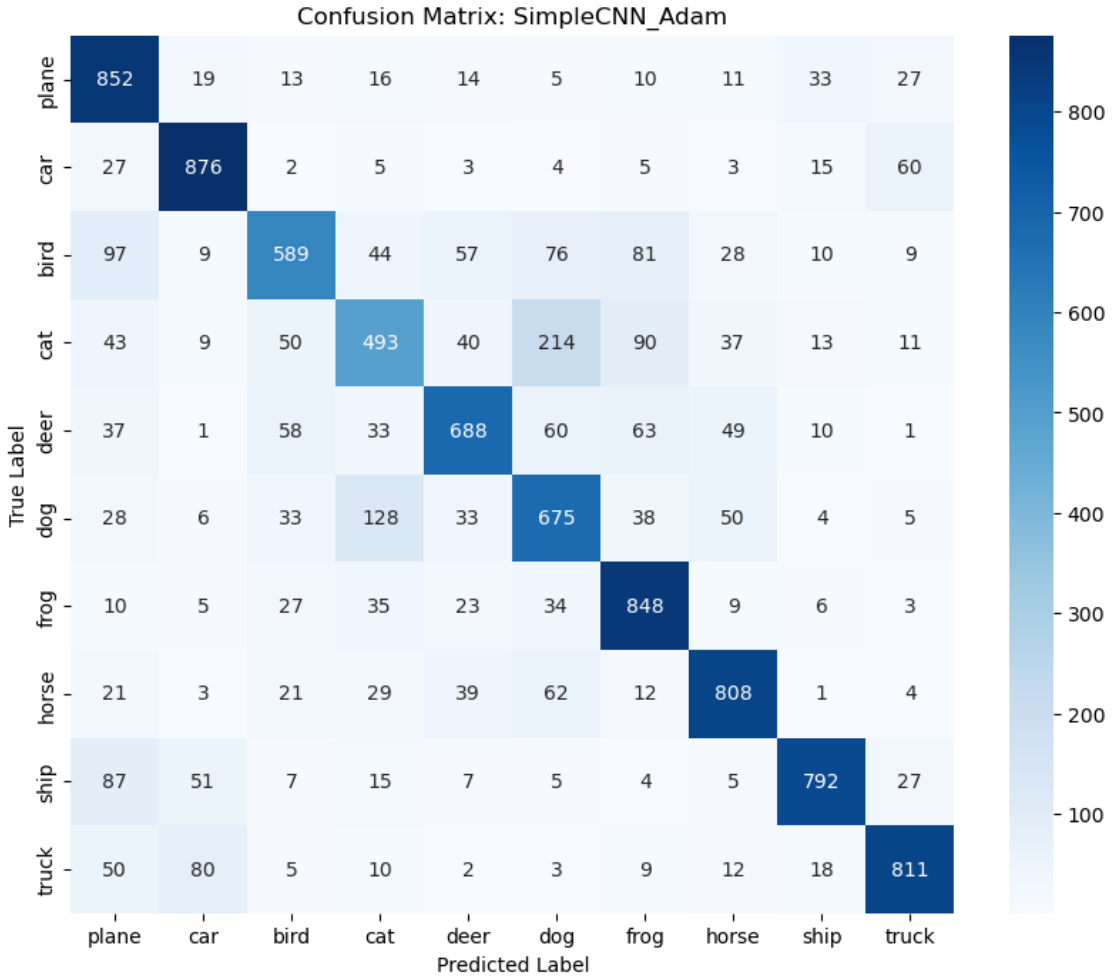

```
  Summary Statistics:
- Best Validation Accuracy: 86.14% (Epoch 19)
- Final Validation Accuracy: 84.29%
- Training Time: 733.79 seconds
- Scheduler Used: ReduceLROnPlateau
Schedular Comparison Table
                      Experiment         Scheduler Best Val Acc (%) Final Val
Acc (%) Time (s) Final LR
             SimpleCNN_Adam_None              None            75.14
75.09   604.44 0.001000
           SimpleCNN_Adam_StepLR            StepLR            72.05
72.01   587.59 0.000010
SimpleCNN_Adam_ReduceLROnPlateau ReduceLROnPlateau            75.21
75.15   598.43 0.001000
             ResNet-18_Adam_None              None            85.10
84.48   642.29 0.001000
           ResNet-18_Adam_StepLR            StepLR            87.13
87.03   768.41 0.000010
ResNet-18_Adam_ReduceLROnPlateau ReduceLROnPlateau            86.14
84.29   733.79 0.001000
```

[ ]:
```python
# ## 7. Analysis and Visualization of Results
if not experiment_results:
    print("No experiment results found. Please run the training loop first.")
else:
    for experiment_name, results in experiment_results.items():

        print(f"Analyzing: {experiment_name}")

        # Instantiate a new model and load the saved weights
        model = results['model_config']['builder']().to(DEVICE)
        model.load_state_dict(results['model_state'])
        model.eval()

        # Plot Training History
        print(f"[Plotting] Training History for {experiment_name}...")
```

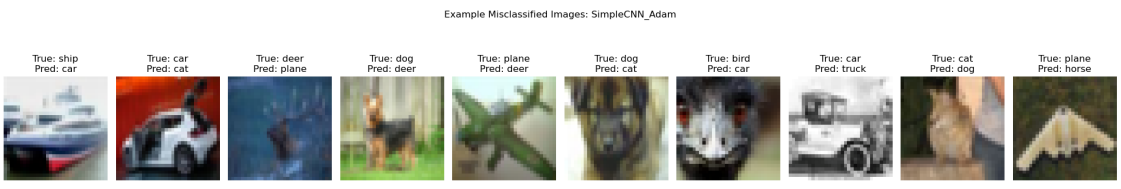
```
        plot_history(results['history'], title=experiment_name)

        # Evaluate final model to get labels and predictions for plots
        print(f"[Evaluating] Final model for {experiment_name}...")
        _, _, all_labels, all_preds = evaluate_model(
            model, test_loader, criterion, DEVICE)

        # Plot Confusion Matrix
        print(f"[Plotting] Confusion Matrix for {experiment_name}...")
        plot_confusion_matrix(all_labels, all_preds, CLASS_NAMES,␣
↪title=experiment_name)

        # Plot Misclassified Images
        print(f"[Plotting] Misclassified Images for {experiment_name}...")
        plot_misclassified(model, test_loader, CLASS_NAMES, DEVICE,␣
↪title=experiment_name, num_images=10)

        # Plot Activation Maps
        print(f"[Plotting] Activation Maps for {experiment_name}...")

        # Define which layers to inspect for each model type
        if results['model_config']['name'] == 'SimpleCNN':
            layers_to_plot = {
                'Layer 1 (conv1)': model.conv1,
                'Layer 2 (conv2)': model.conv2
            }
        elif results['model_config']['name'] == 'ResNet-18':
            layers_to_plot = {
                'Layer 1 (ResBlock)': model.layer1,
                'Layer 2 (ResBlock)': model.layer2
            }

        plot_activation_maps(model, layers_to_plot, test_loader, DEVICE,␣
↪title=experiment_name, num_images=5)
```

```
Analyzing: SimpleCNN_Adam
[Plotting] Training History for SimpleCNN_Adam...
```

[Evaluating] Final model for SimpleCNN_Adam…
[Plotting] Confusion Matrix for SimpleCNN_Adam…



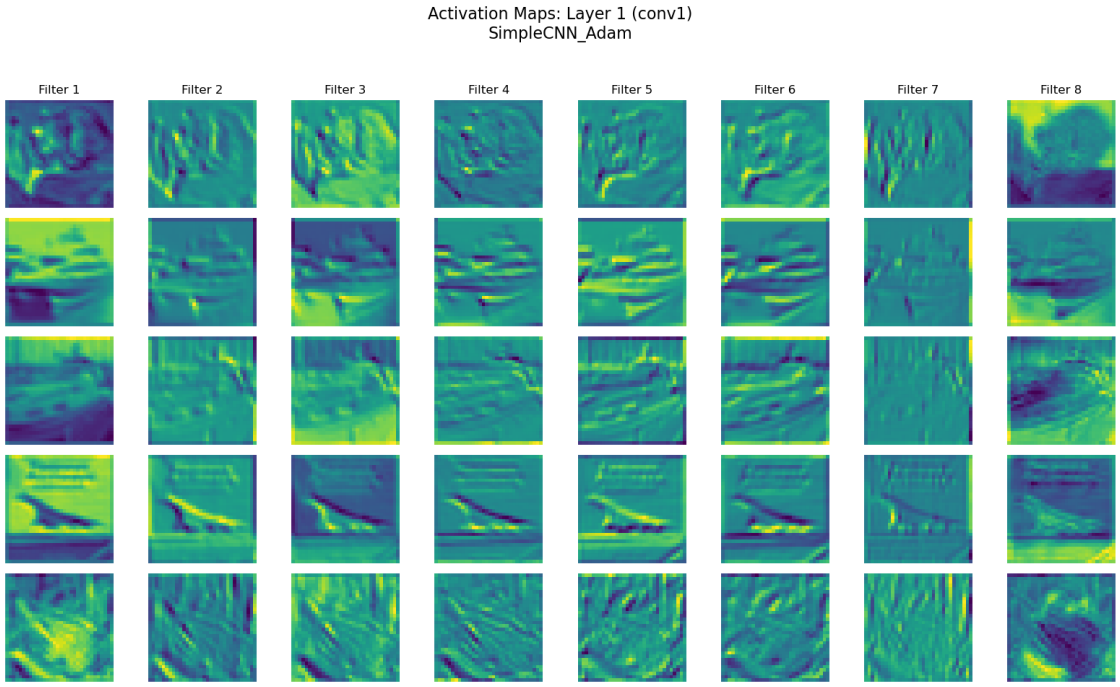
[Plotting] Misclassified Images for SimpleCNN_Adam…

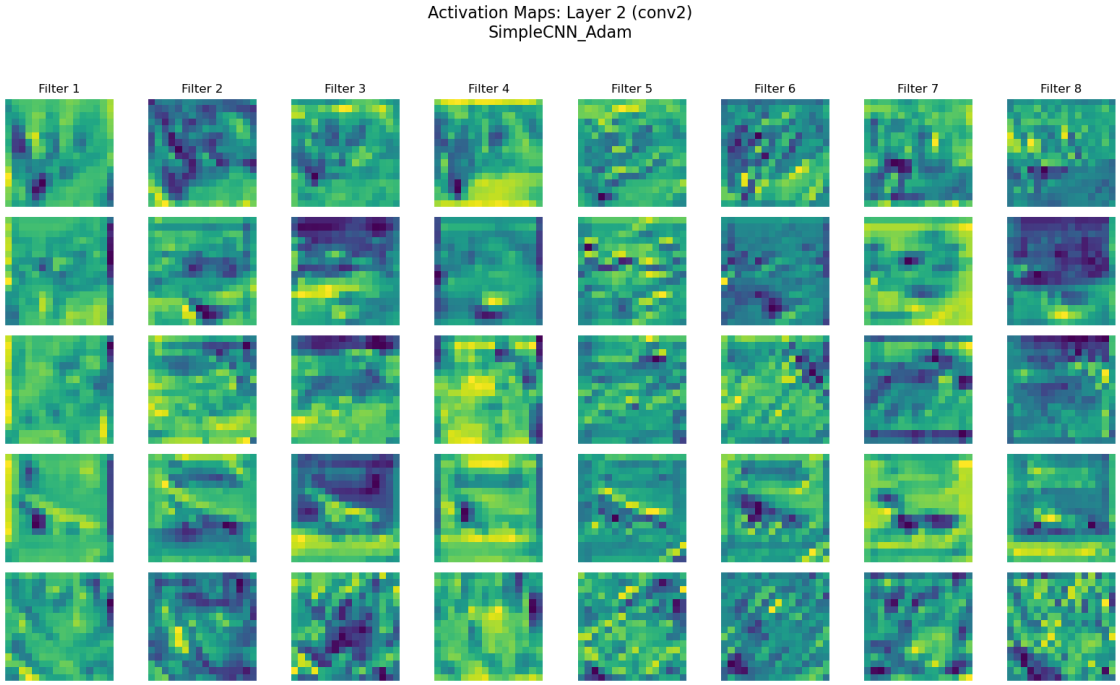


[Plotting] Activation Maps for SimpleCNN_Adam…

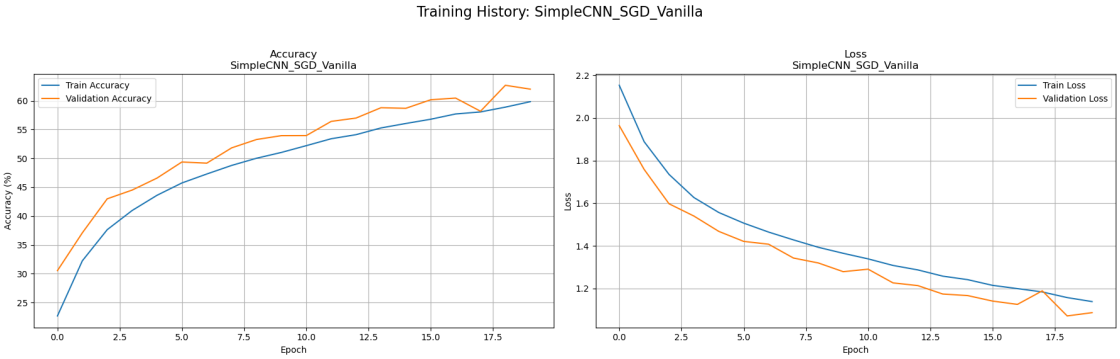
Original Images
SimpleCNN_Adam

```
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at
0x000001D085FE1FC0>
Traceback (most recent call last):
  File "c:\Users\mfaha\miniconda3\envs\cuda_env\lib\site-
packages\torch\utils\data\dataloader.py", line 1604, in __del__
    self._shutdown_workers()
  File "c:\Users\mfaha\miniconda3\envs\cuda_env\lib\site-
packages\torch\utils\data\dataloader.py", line 1562, in _shutdown_workers
    if self._persistent_workers or self._workers_status[worker_id]:
AttributeError: '_MultiProcessingDataLoaderIter' object has no attribute
'_workers_status'
```

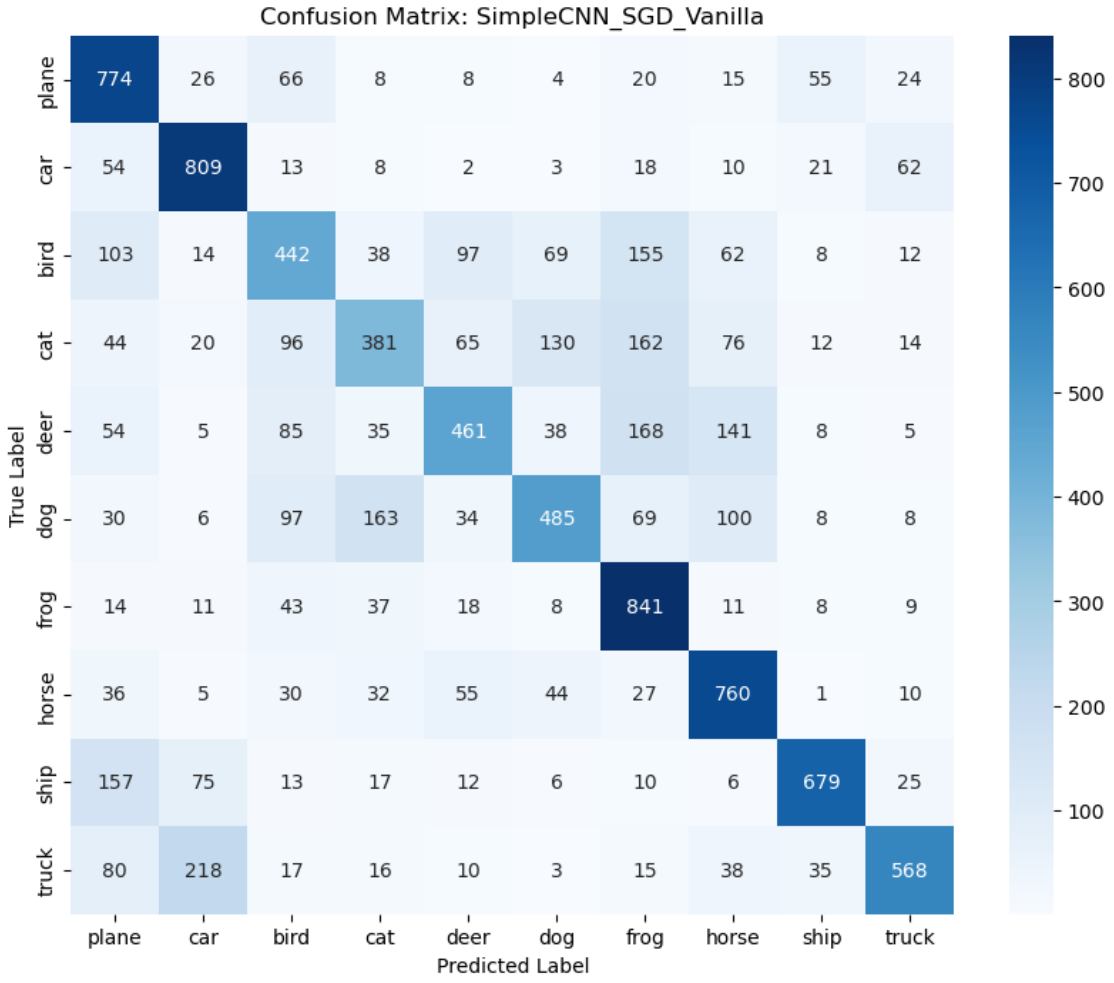




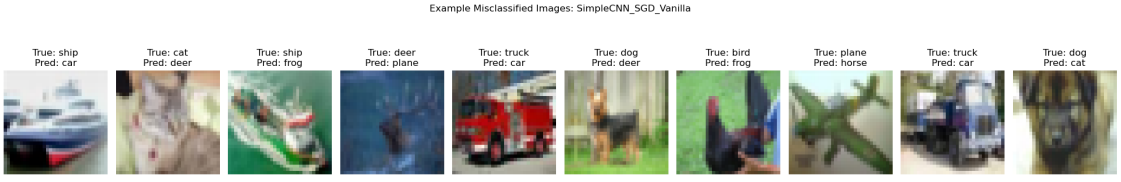
Analyzing: SimpleCNN_SGD_Vanilla
[Plotting] Training History for SimpleCNN_SGD_Vanilla…



[Evaluating] Final model for SimpleCNN_SGD_Vanilla…
[Plotting] Confusion Matrix for SimpleCNN_SGD_Vanilla…

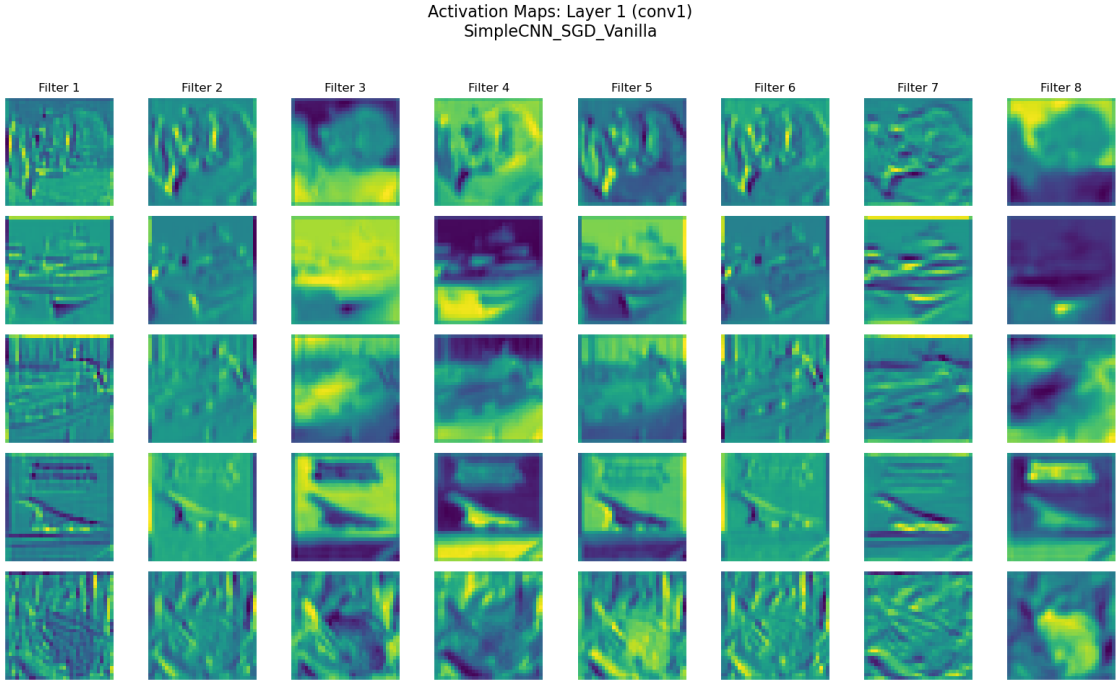


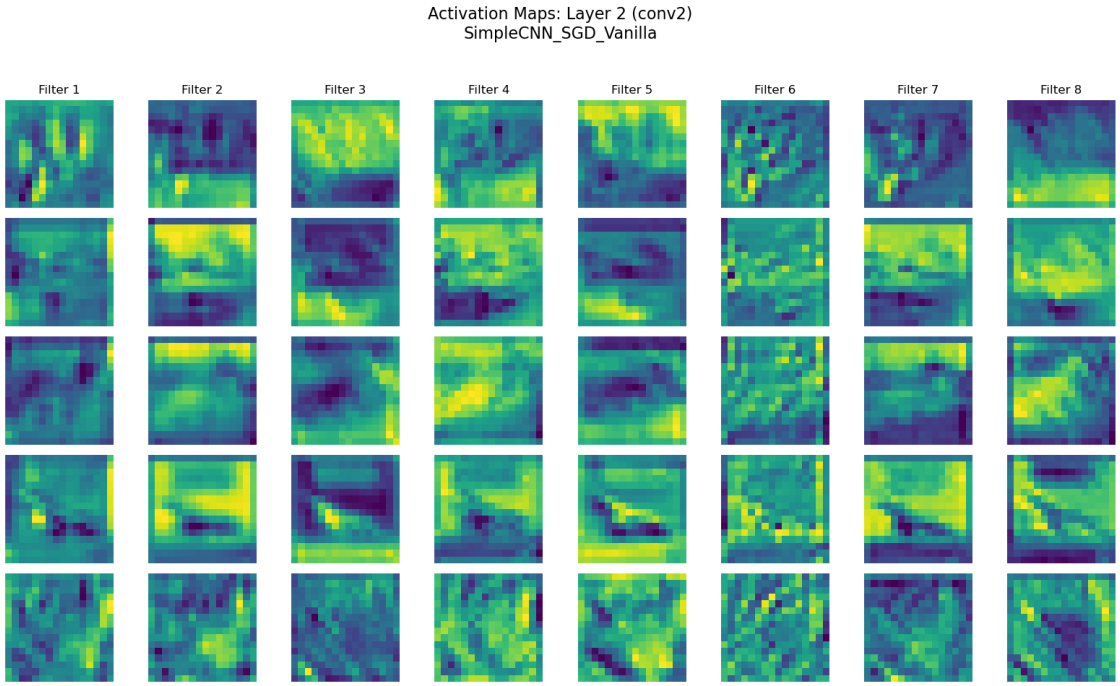
[Plotting] Misclassified Images for SimpleCNN_SGD_Vanilla…



[Plotting] Activation Maps for SimpleCNN_SGD_Vanilla…

Original Images
SimpleCNN_SGD_Vanilla

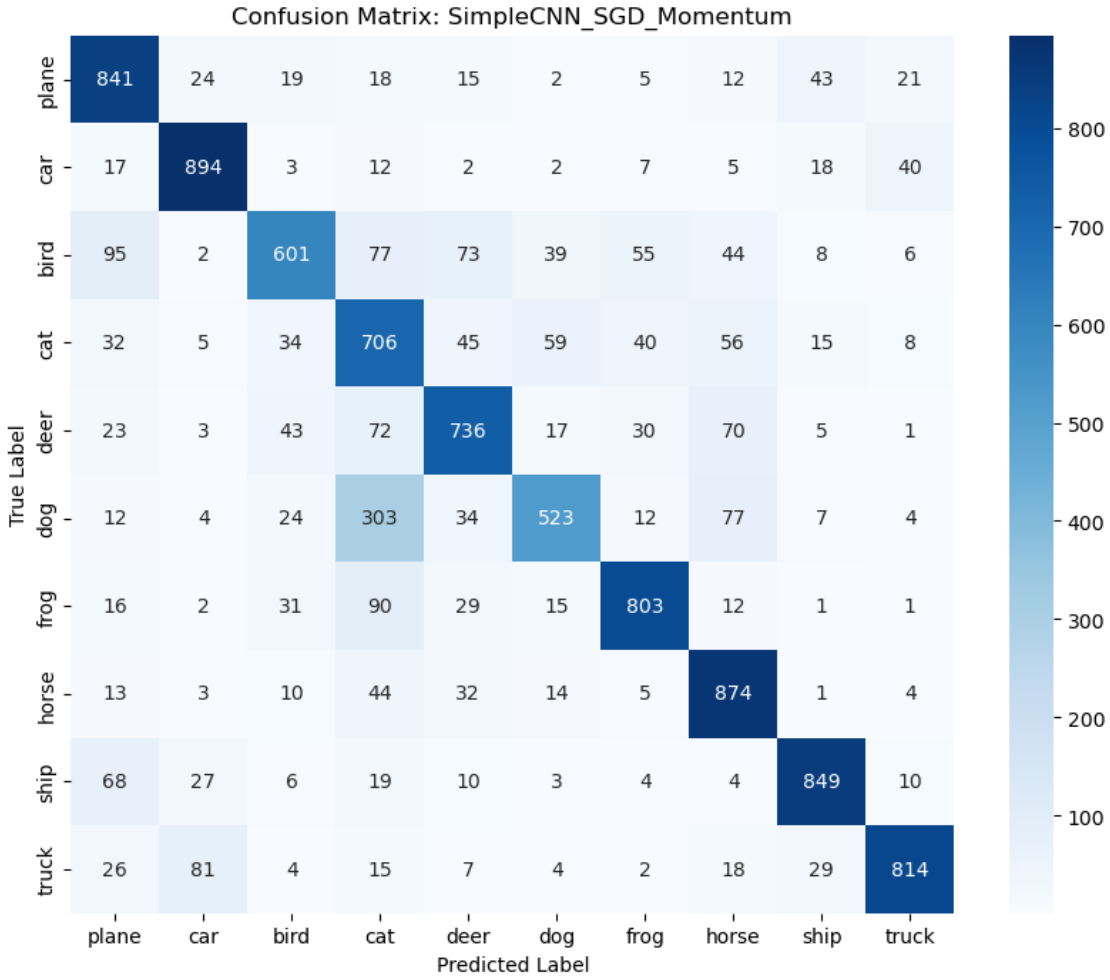




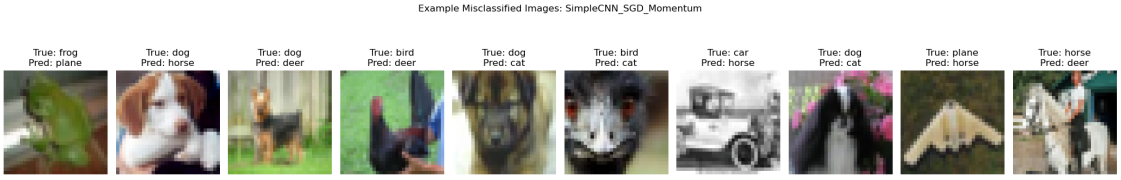

```
Analyzing: SimpleCNN_SGD_Momentum
```

[Plotting] Training History for SimpleCNN_SGD_Momentum...

[Evaluating] Final model for SimpleCNN_SGD_Momentum...
[Plotting] Confusion Matrix for SimpleCNN_SGD_Momentum...

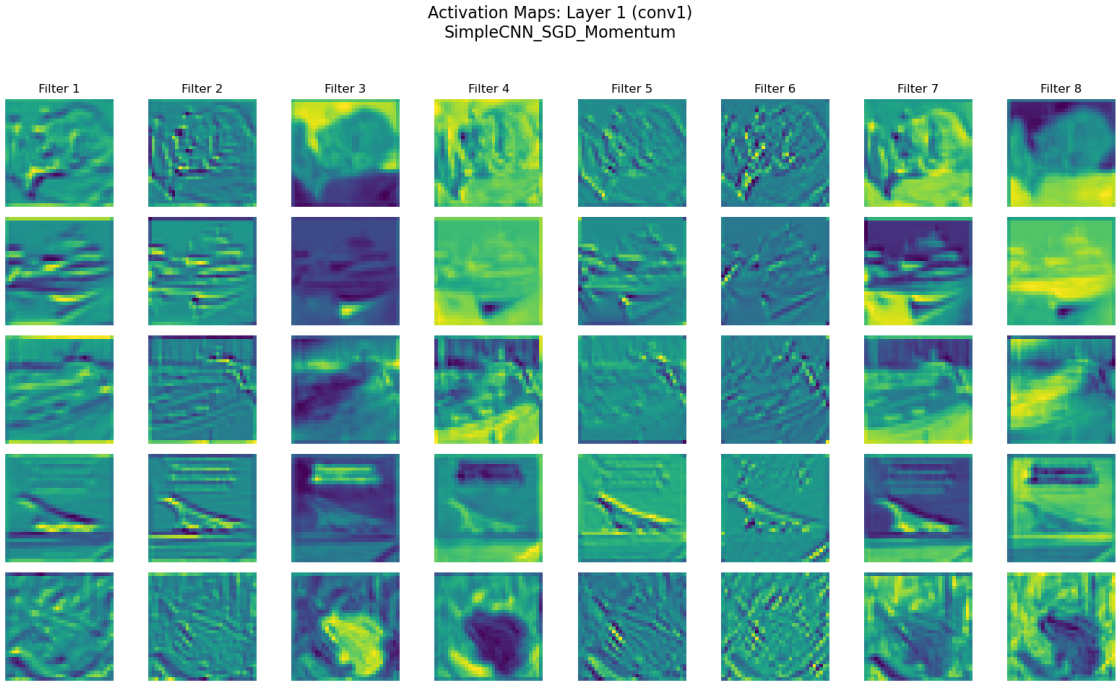


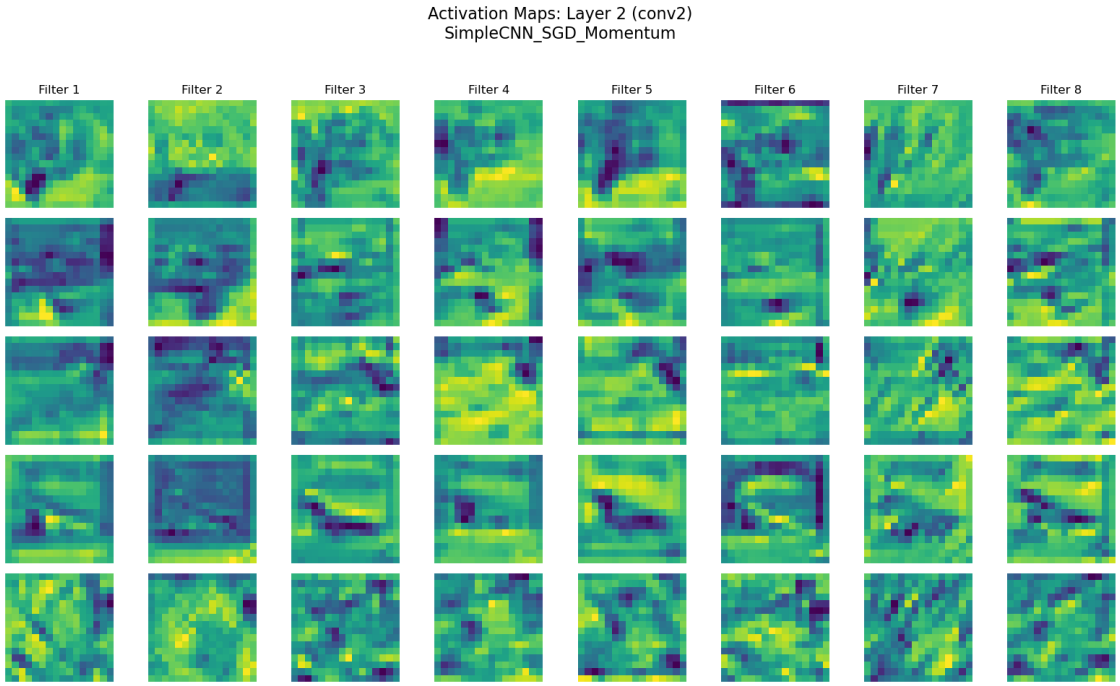
[Plotting] Misclassified Images for SimpleCNN_SGD_Momentum…



[Plotting] Activation Maps for SimpleCNN_SGD_Momentum…

Original Images
SimpleCNN_SGD_Momentum

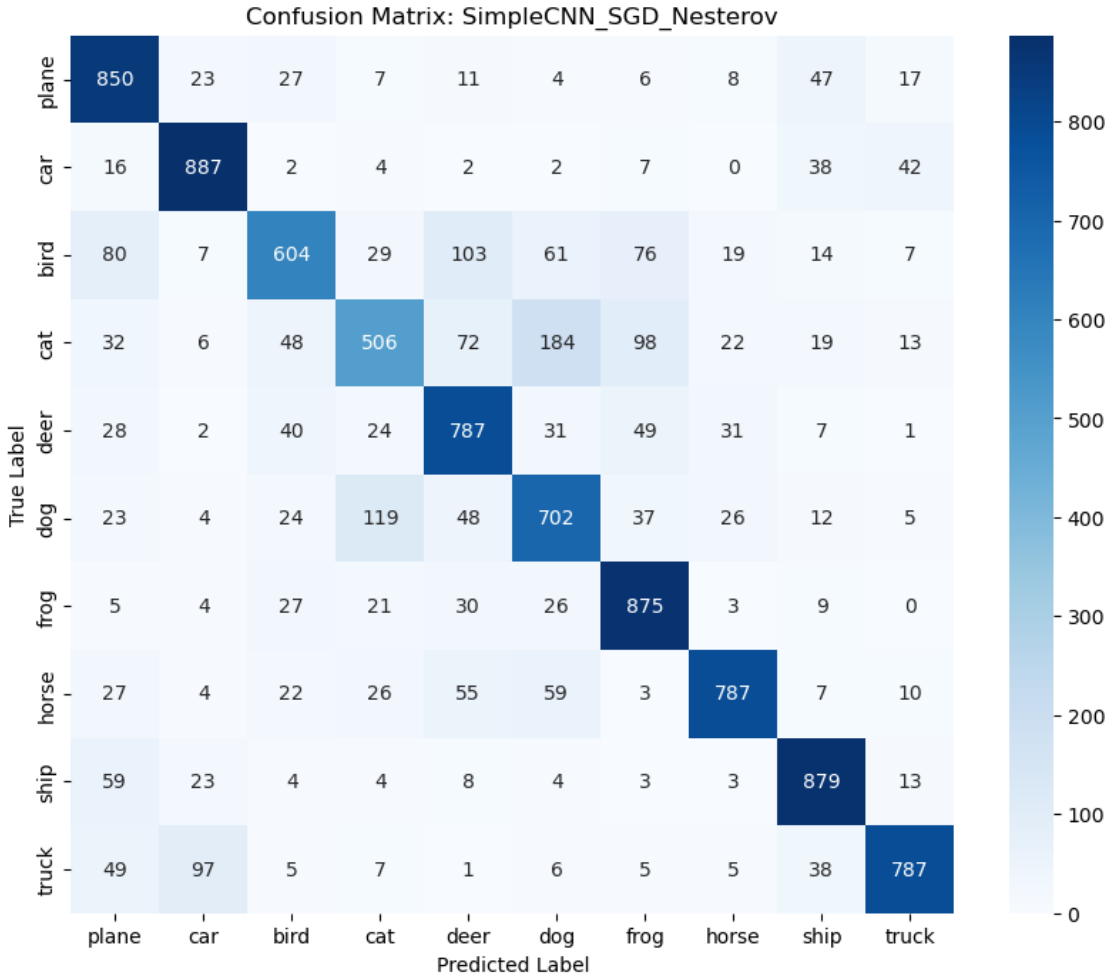




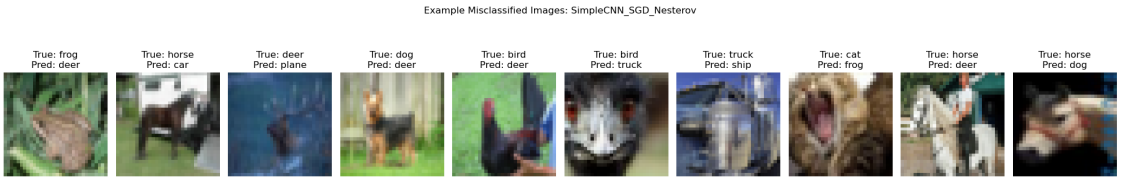

```
Analyzing: SimpleCNN_SGD_Nesterov
```

[Plotting] Training History for SimpleCNN_SGD_Nesterov…

[Evaluating] Final model for SimpleCNN_SGD_Nesterov…
[Plotting] Confusion Matrix for SimpleCNN_SGD_Nesterov…

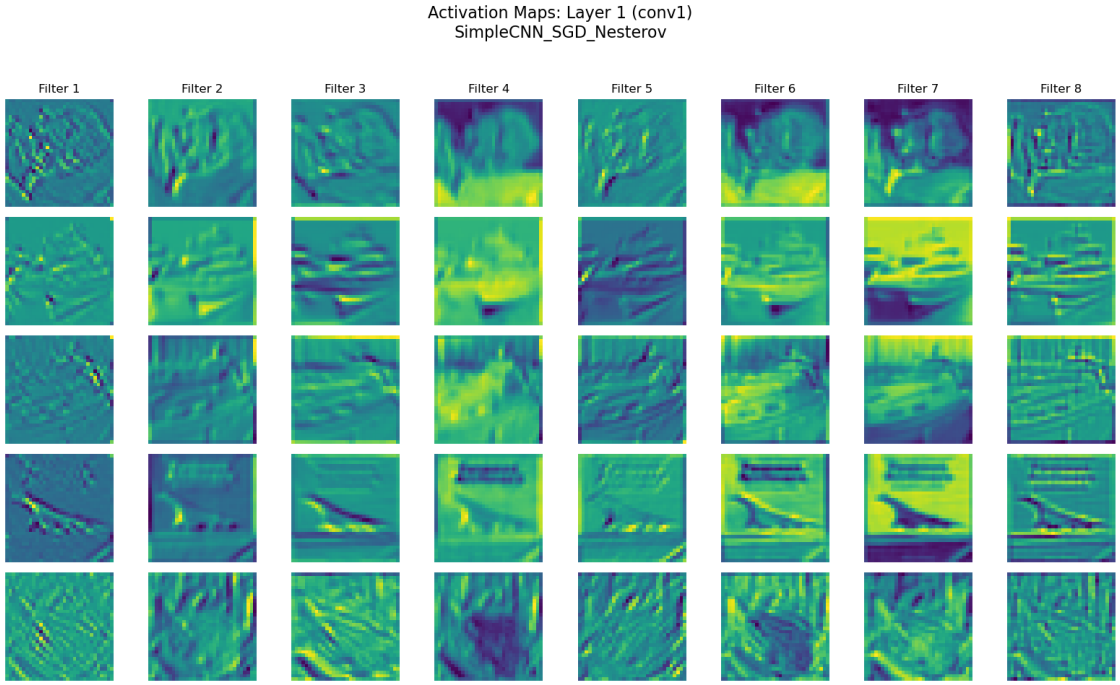


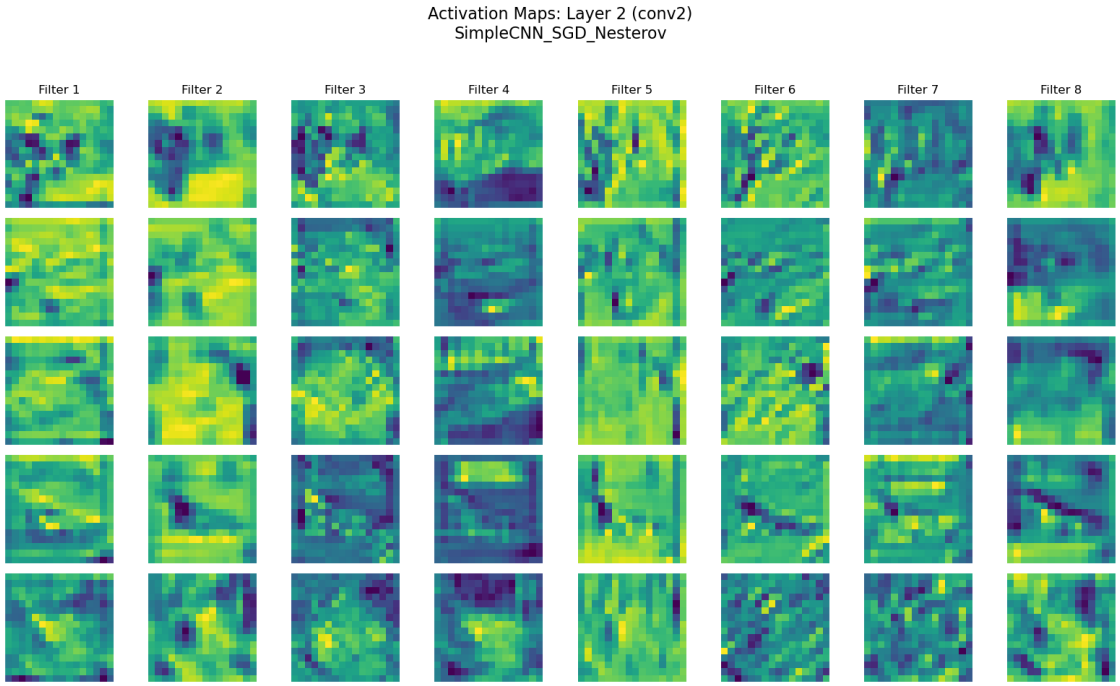
[Plotting] Misclassified Images for SimpleCNN_SGD_Nesterov…



[Plotting] Activation Maps for SimpleCNN_SGD_Nesterov…

Original Images
SimpleCNN_SGD_Nesterov

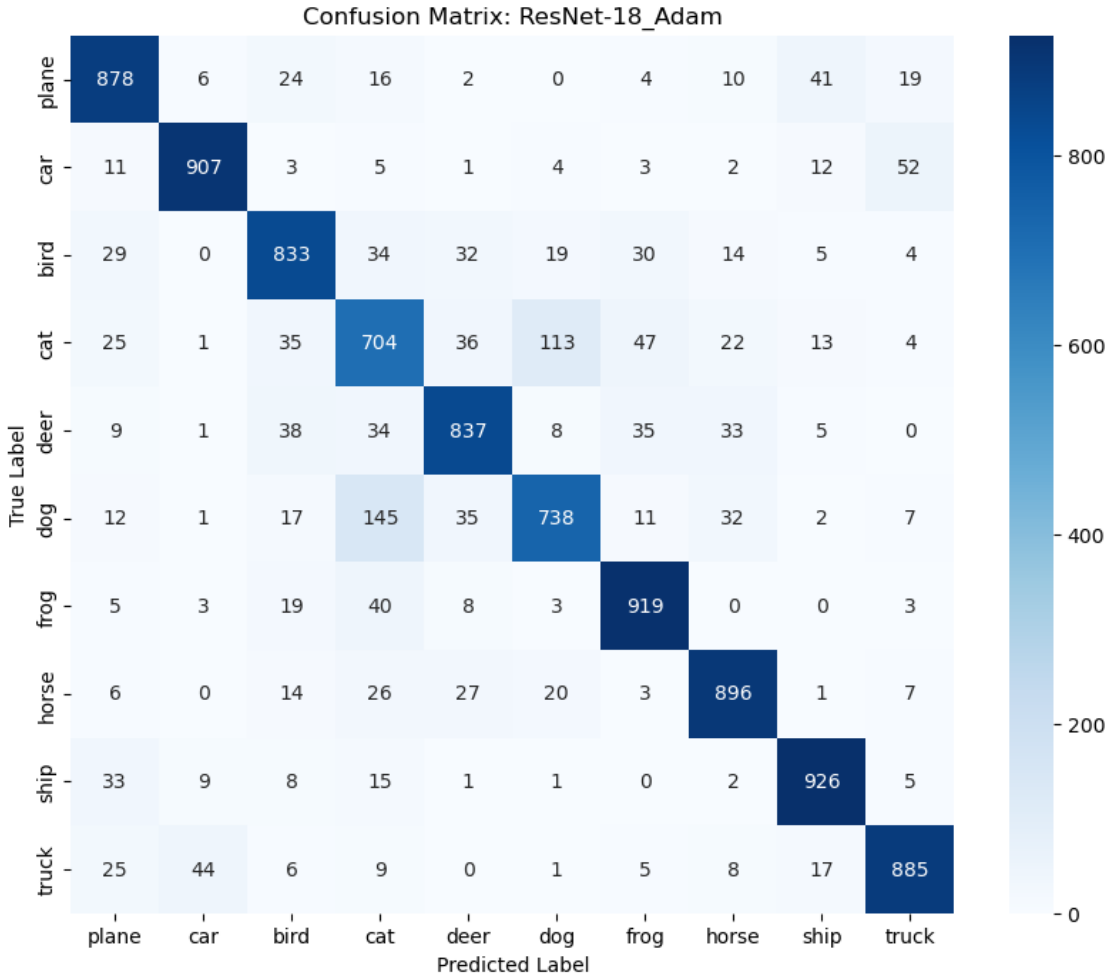




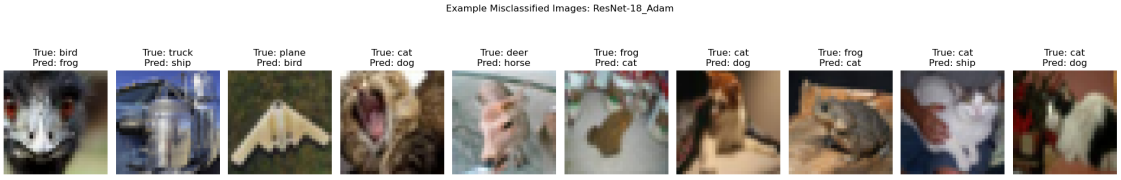

```
Analyzing: ResNet-18_Adam
```

[Plotting] Training History for ResNet-18_Adam…

[Evaluating] Final model for ResNet-18_Adam…
[Plotting] Confusion Matrix for ResNet-18_Adam…

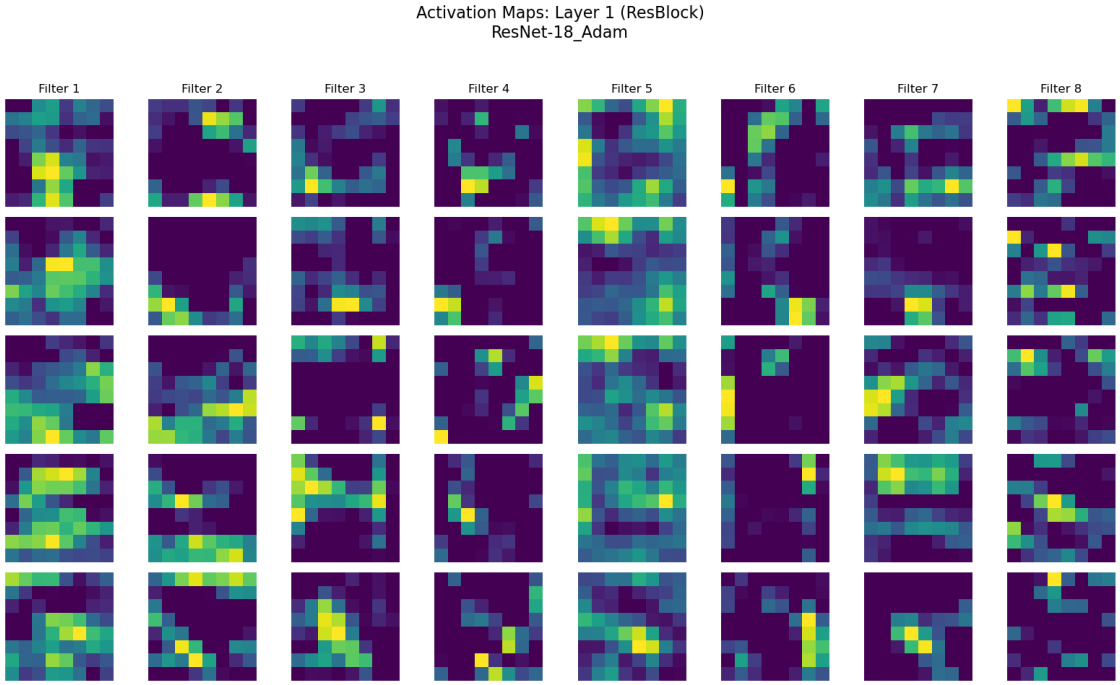


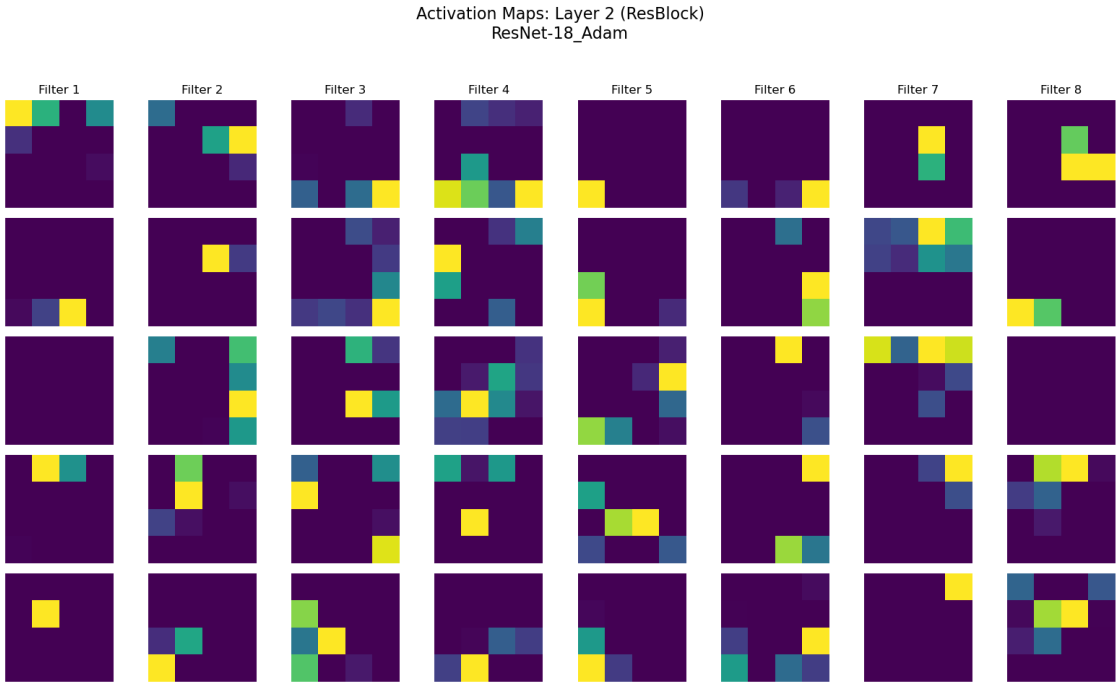
[Plotting] Misclassified Images for ResNet-18_Adam…



[Plotting] Activation Maps for ResNet-18_Adam…

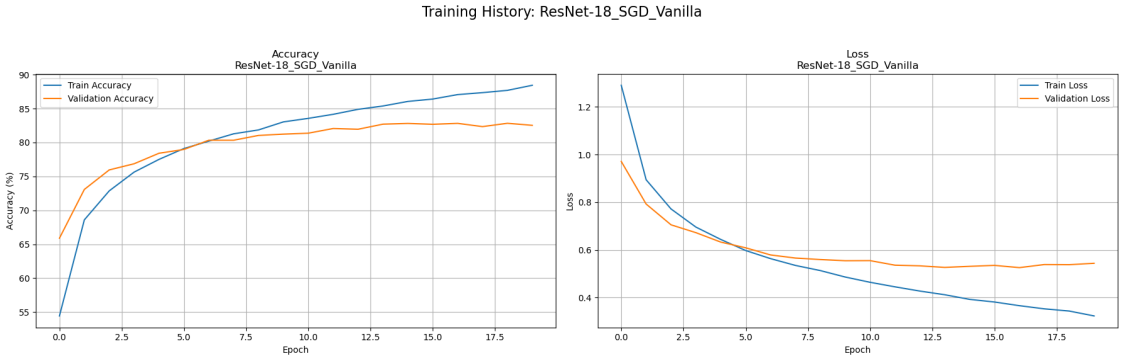
Original Images
ResNet-18_Adam



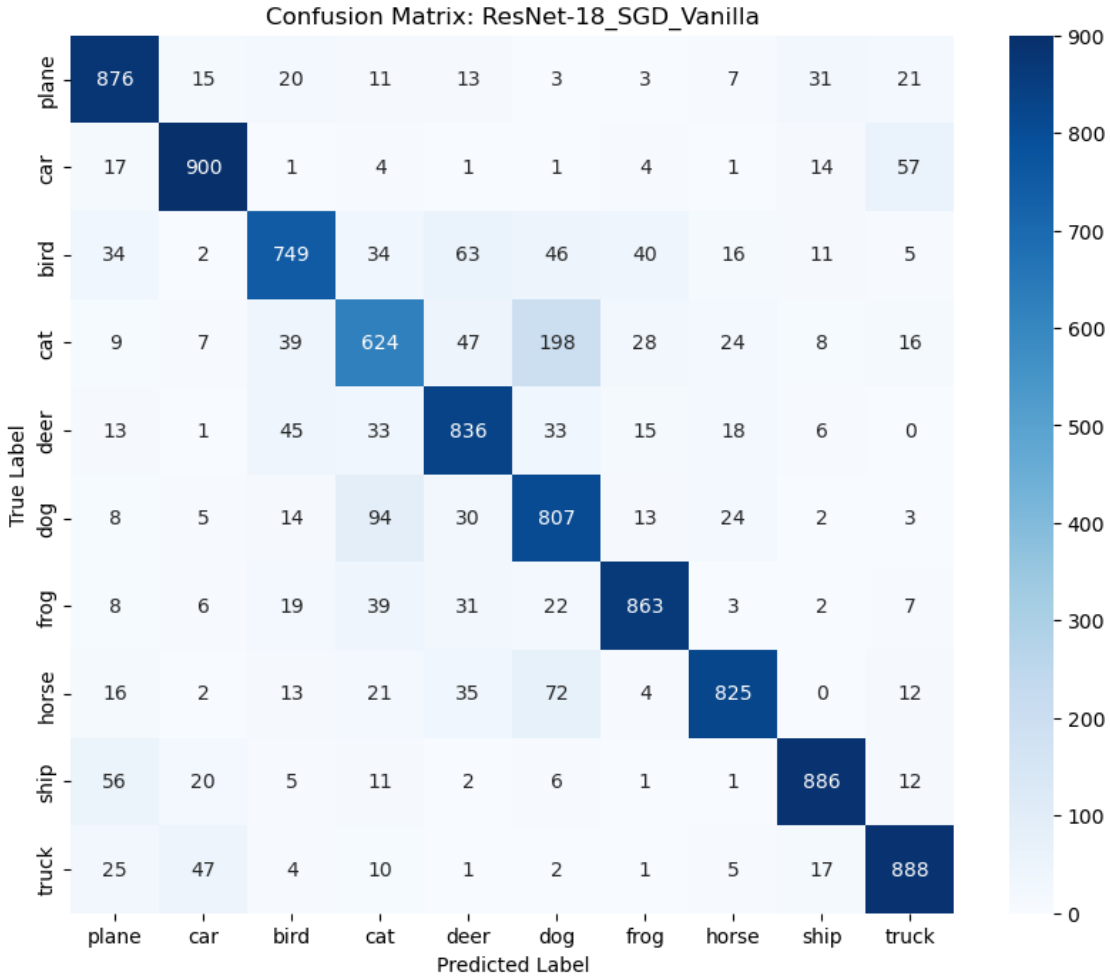



```
Analyzing: ResNet-18_SGD_Vanilla
```

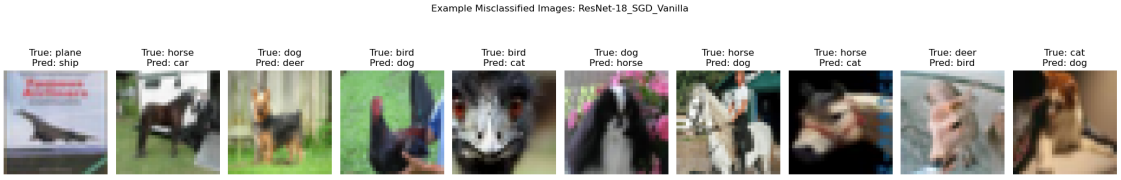
[Plotting] Training History for ResNet-18_SGD_Vanilla…



[Evaluating] Final model for ResNet-18_SGD_Vanilla…
[Plotting] Confusion Matrix for ResNet-18_SGD_Vanilla…

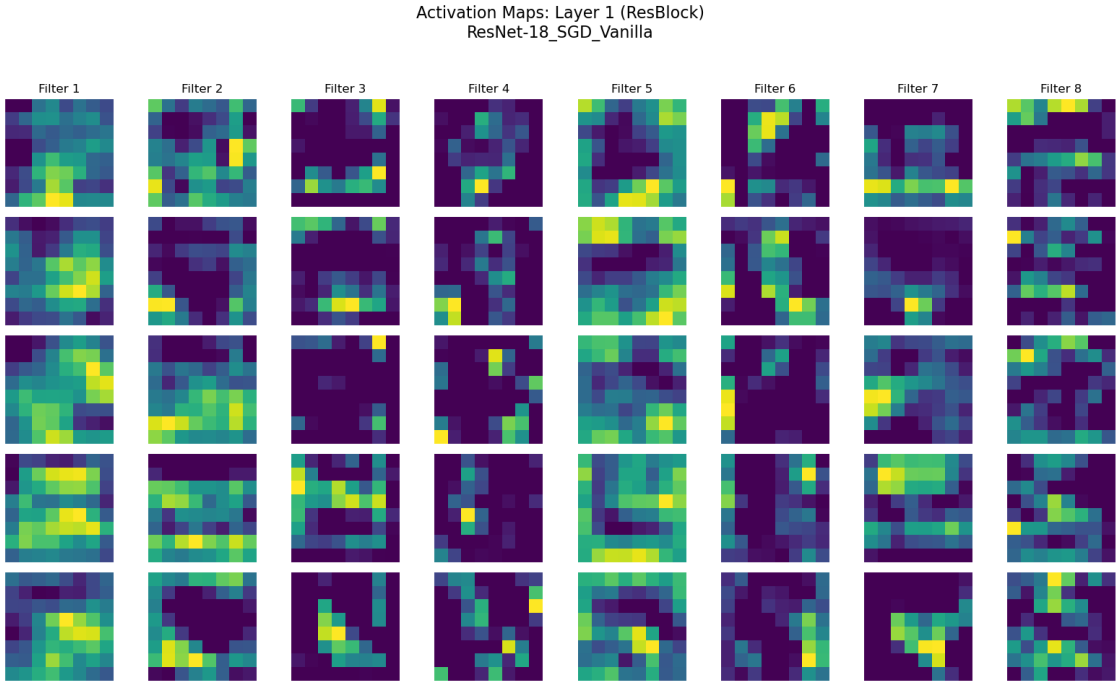


[Plotting] Misclassified Images for ResNet-18_SGD_Vanilla…



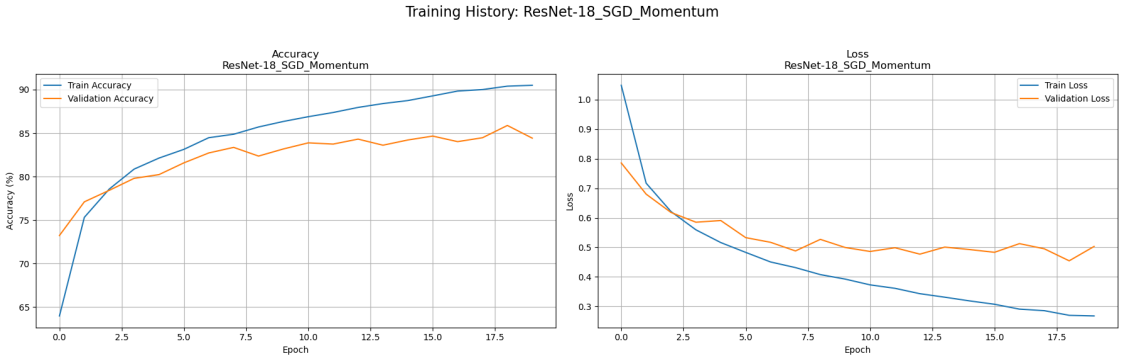
[Plotting] Activation Maps for ResNet-18_SGD_Vanilla…

Original Images
ResNet-18_SGD_Vanilla

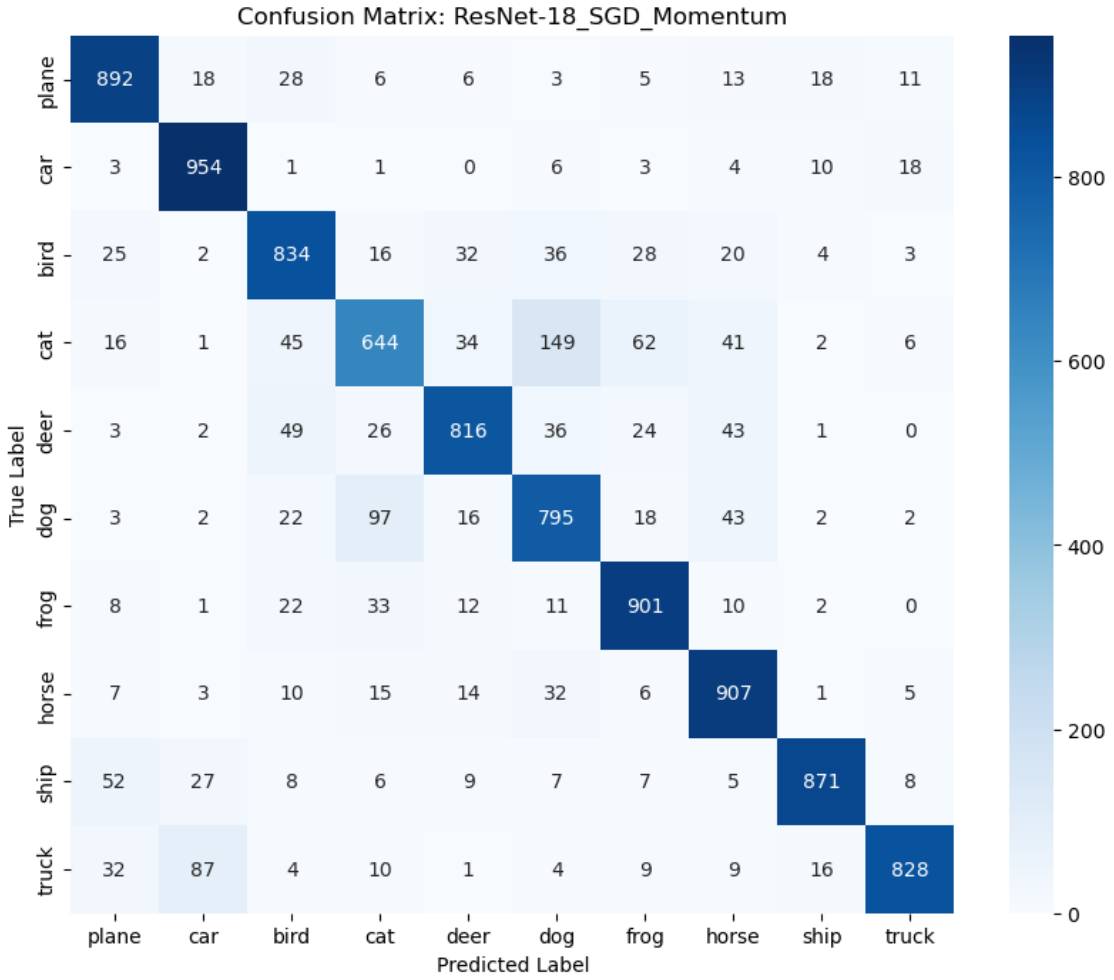



```
Analyzing: ResNet-18_SGD_Momentum
```

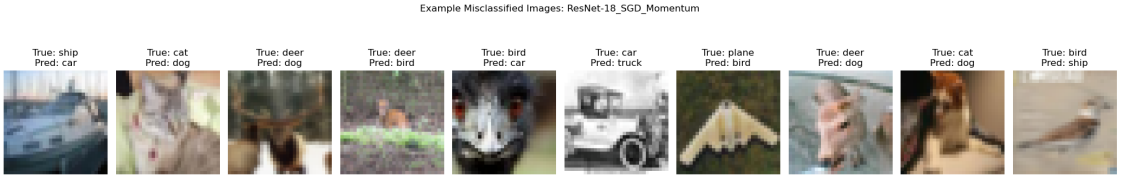
[Plotting] Training History for ResNet-18_SGD_Momentum…



[Evaluating] Final model for ResNet-18_SGD_Momentum…
[Plotting] Confusion Matrix for ResNet-18_SGD_Momentum…

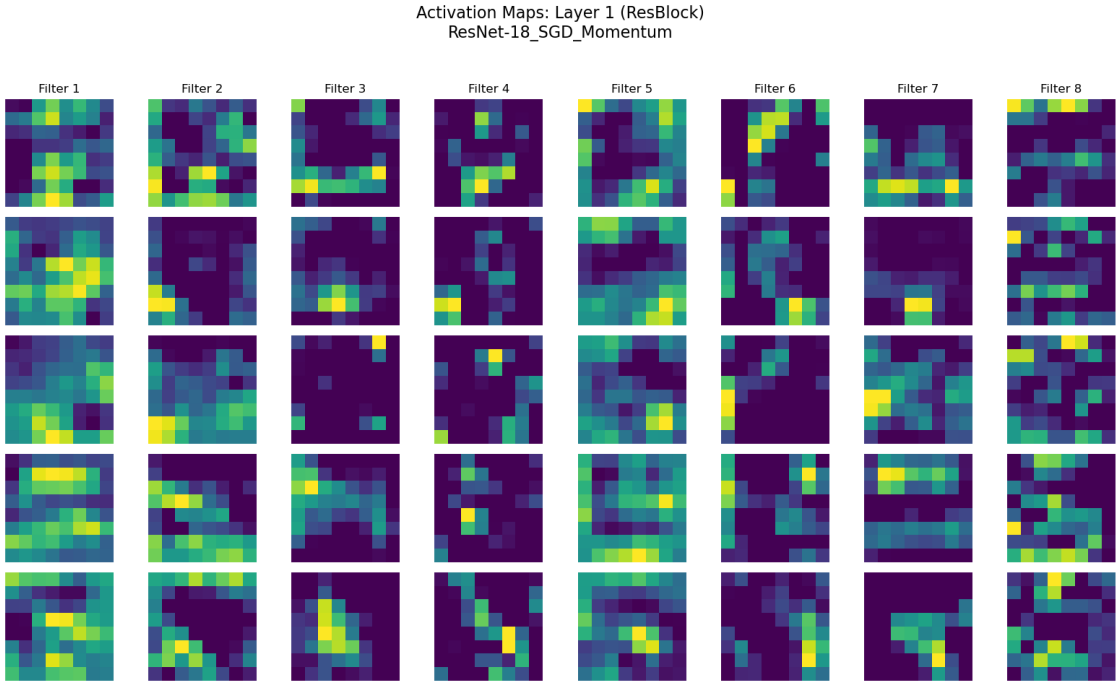


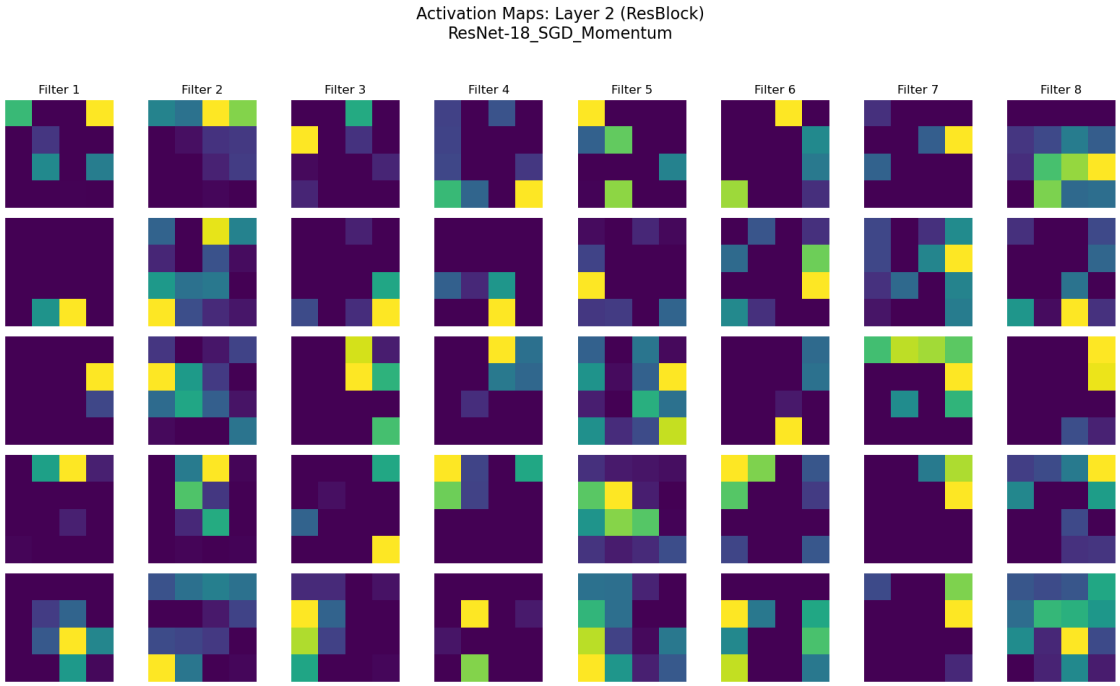
[Plotting] Misclassified Images for ResNet-18_SGD_Momentum…



[Plotting] Activation Maps for ResNet-18_SGD_Momentum…

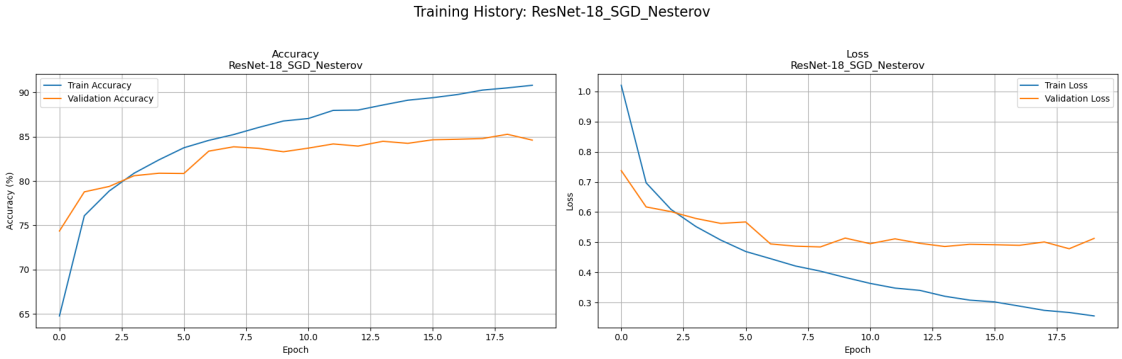
Original Images
ResNet-18_SGD_Momentum



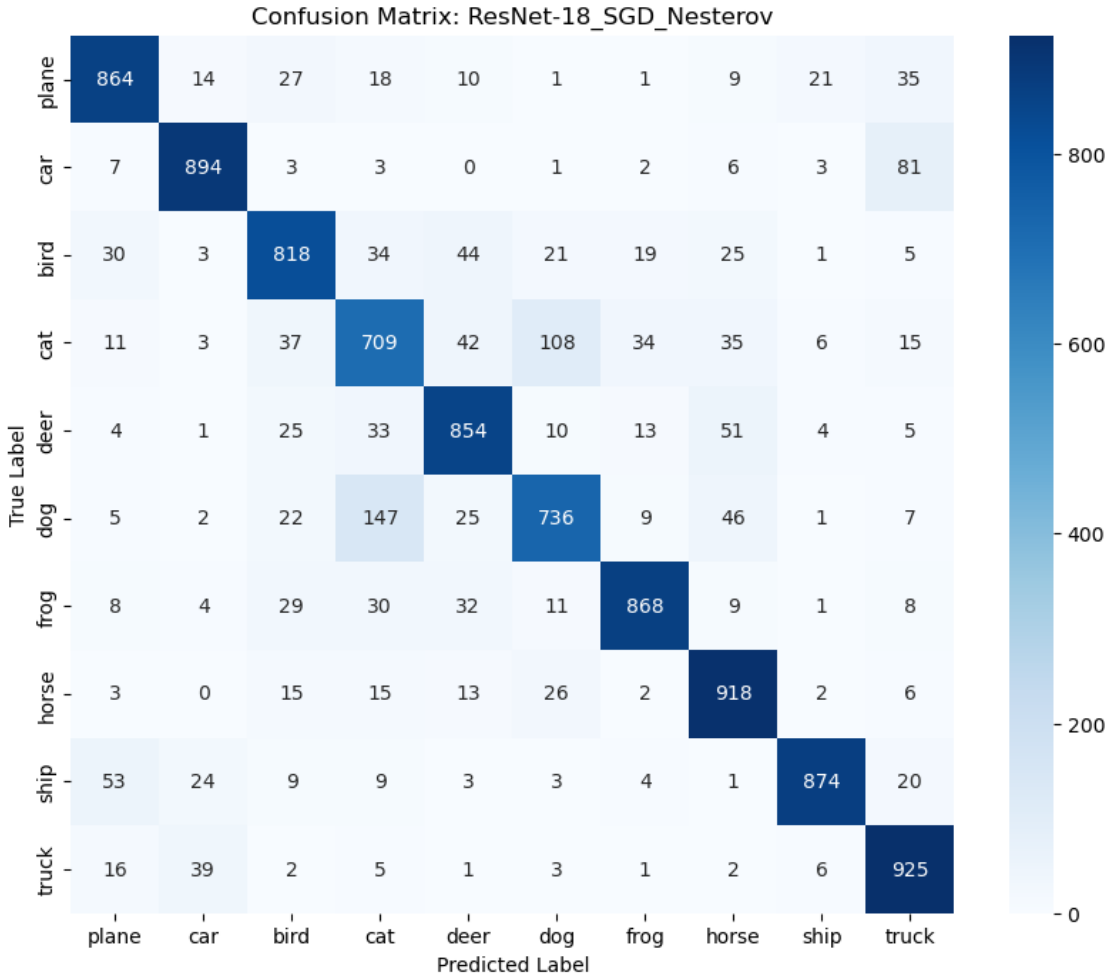



```
Analyzing: ResNet-18_SGD_Nesterov
```

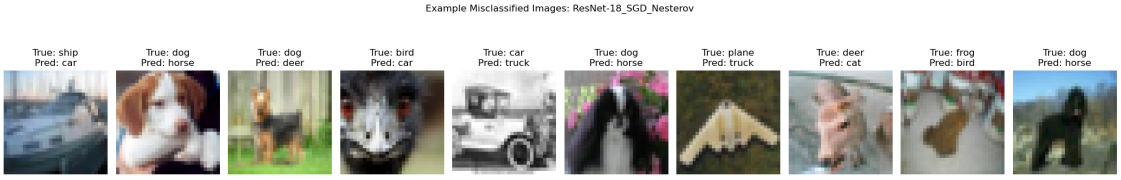
[Plotting] Training History for ResNet-18_SGD_Nesterov…



[Evaluating] Final model for ResNet-18_SGD_Nesterov…
[Plotting] Confusion Matrix for ResNet-18_SGD_Nesterov…

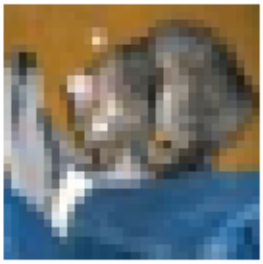


[Plotting] Misclassified Images for ResNet-18_SGD_Nesterov…

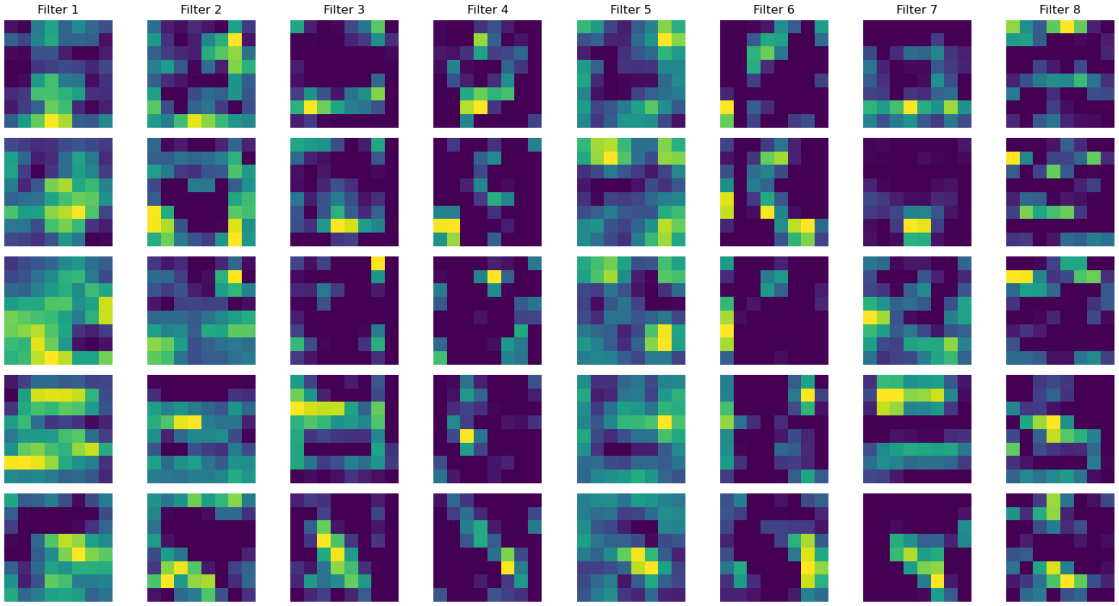


[Plotting] Activation Maps for ResNet-18_SGD_Nesterov…

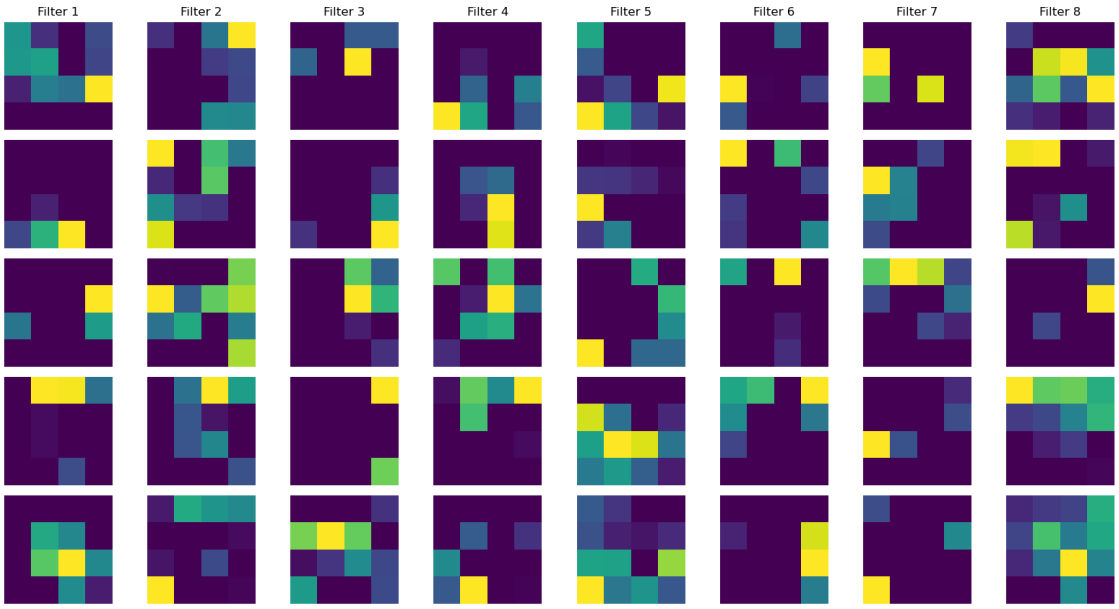
Original Images
ResNet-18_SGD_Nesterov



Activation Maps: Layer 1 (ResBlock)
ResNet-18_SGD_Nesterov



Activation Maps: Layer 2 (ResBlock)
ResNet-18_SGD_Nesterov



0.2 Discussion

- **Best Overall Performance:** ResNet-18 + Adam + StepLR → **87.13% accuracy**
- **Best SimpleCNN Configuration:** SimpleCNN + SGD Nesterov → **76.86% accuracy**

0.3 1. Architecture Comparison: SimpleCNN vs ResNet-18

0.3.1 Performance Summary

Architecture	Best Accuracy	Parameters	Training Time	Start Acc (Epoch 1)
SimpleCNN	76.86%	~0.15M	~515s	23–41%
ResNet-18	87.13%	~11.2M	~626s	64–66%
Difference**	+10.27%	75× larger	+22% slower	+30% higher start

0.3.2 Key Findings

ResNet-18 Advantages

- **Transfer Learning:** Pre-trained ImageNet weights give it strong initial representations.
- **Residual Connections:** Prevent vanishing gradients, allowing deeper training.
- **Rich Feature Hierarchies:** Learns semantic and abstract patterns beyond simple edges.

SimpleCNN Advantages

- **Efficiency:** 75× fewer parameters and 22% faster training.
- **Simplicity:** Easy to deploy on edge or mobile devices.
- **Interpretability:** Activation maps clearly show basic edge and texture features.

  ResNet-18 offers strong performance through transfer learning and depth, while SimpleCNN excels in efficiency and interpretability.

0.4 2. Optimizer Comparison

0.4.1 Performance Table

Model	Optimizer	Final Val Acc	Training Time	Rank
SimpleCNN	SGD Nesterov	76.86%	496s	1st
SimpleCNN	Adam	75.77%	542s	2nd
SimpleCNN	SGD Momentum	74.70%	472s	3rd
SimpleCNN	SGD Vanilla	61.84%	472s	4th
ResNet-18	Adam	85.45%	525s	1st
ResNet-18	SGD Nesterov	84.63%	616s	2nd
ResNet-18	SGD Momentum	84.32%	541s	3rd
ResNet-18	SGD Vanilla	82.87%	477s	4th

0.4.2 Key Insights

- **Momentum is Essential:**
  SGD with momentum boosts accuracy by **12–15%** over vanilla SGD.
- **Architecture–Optimizer Interaction:**
  - *SimpleCNN → Nesterov*: Look-ahead gradient helps shallow networks (+1.09% over Adam).
  - *ResNet-18 → Adam*: Adaptive learning rates complement transfer learning best.
- **Adam Characteristics:**
  - Fastest initial convergence
  - Robust to hyperparameter variations
  - Excellent for transfer learning
- **Nesterov vs Momentum:**
  - Minor difference in deep models
  - ~2% benefit for shallow networks (SimpleCNN)

0.5 3. Learning Rate Scheduler Analysis

All tests here use **Adam optimizer** to isolate scheduler effects.

0.5.1 Performance Summary

Model	Scheduler	Best Val Acc	Δ Improvement	Final LR
SimpleCNN	ReduceLROnPlateau	75.21%	+0.07%	0.001000
SimpleCNN	None	75.14%	baseline	0.001000
SimpleCNN	StepLR	72.05%	-3.08%	0.000010
ResNet-18	StepLR	87.13%	+2.03%	0.000010
ResNet-18	ReduceLROnPlateau	86.14%	+1.04%	0.001000
ResNet-18	None	85.10%	baseline	0.001000

0.5.2 Critical Findings

- **Architecture-Dependent Effects:**
  - **StepLR harms SimpleCNN (-3.08%)** LR drops too early, halting learning.
  - **StepLR benefits ResNet-18 (+2.03%)** Pre-trained weights fine-tune efficiently at lower LR.
- **ReduceLROnPlateau as Safe Default:**
  - Adapts to actual loss trends.

- Works consistently across architectures.
- More conservative than StepLR, but safer when tuning unknown models.

0.6 4. Confusion Matrix Insights

0.6.1 SimpleCNN Common Errors

- **Cat Dog:** Similar fur textures, shallow features.
- **Car Truck:** Overlapping shape features and viewing angles.
- **Bird Airplane:** Silhouette confusion due to limited feature depth.

0.6.2 ResNet-18 Errors

- More **subtle and context-based mistakes**.
- Stronger **animal discrimination**.
- Errors mostly in **ambiguous samples**, not model weakness.

Conclusion:
SimpleCNN makes *obvious category confusions*, while ResNet-18's errors are *semantically harder cases*.

0.7 5. Overall Results

0.7.1 Best Configurations

Use Case	Model	Optimizer	Scheduler	Accuracy
Production (Max Accuracy)	ResNet-18 (pre-trained)	Adam (lr=0.001)	StepLR (step=7, =0.1)	87.13%
Edge Devices (Efficiency)	SimpleCNN	SGD Nesterov (lr=0.01, m=0.9)	None / ReduceLROnPlateau	76.86%
Quick Prototyping	Any	Adam (lr=0.001)	ReduceLROnPlateau	—

0.8 Key Takeaways

- **Architecture matters most:** ~10% accuracy gap between SimpleCNN and ResNet-18.
- **Momentum is essential:** +12–15% over vanilla SGD.
- **Transfer learning is powerful:** ResNet-18 starts at 64% vs 23% (epoch 1).
- **Schedulers aren't universal:** StepLR helps deep models but harms shallow ones.

- **Adam is the best default** for transfer learning and stable convergence.