In [ ]:
```
import numpy as np
import pandas as pd
import os
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader, random_split, TensorDataset
from torch.optim.lr_scheduler import StepLR, MultiStepLR
from PIL import Image
import torch.optim.lr_scheduler as lr_scheduler
import matplotlib.pyplot as plt

# auto. choose CPU or GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Function to load CIFAR-10 dataset
def load_cifar_batch(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

# Specify the directory containing CIFAR-10 batches
cifar10_dir = '/content/'
# Load metadata (labels)
meta_data_dict = load_cifar_batch(os.path.join(cifar10_dir, 'batches.meta'))
label_names = [label.decode('utf-8') for label in meta_data_dict[b'label_names']]

# Load training data
train_data = []
train_labels = []
for i in range(1, 6):
    batch = load_cifar_batch(os.path.join(cifar10_dir, f'data_batch_{i}'))
    train_data.append(batch[b'data'])
    train_labels += batch[b'labels']

train_data = np.vstack(train_data).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)  # Convert to HWC format
train_labels = np.array(train_labels)

# Data augmentation and normalization
transform = transforms.Compose([
    transforms.ToPILImage(),  # Convert numpy array to PIL Image
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness = 0.1,contrast = 0.1,saturation = 0.1),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomAdjustSharpness(sharpness_factor = 2,p = 0.2),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261)),
    transforms.RandomErasing(p=0.2,scale=(0.02, 0.1),value=1.0, inplace=False)
])

# Convert to TensorDataset and apply transformations
class CustomCIFAR10Dataset(torch.utils.data.Dataset):
    def __init__(self, images, labels, transform=None):
        self.images = images
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img = self.images[idx]
        label = self.labels[idx]

        if self.transform:
            img = self.transform(img)

        return img, label
```

```
train_dataset = CustomCIFAR10Dataset(train_data, train_labels, transform=transform)

# Split into training and validation sets
#train_size = int(0.9 * len(train_dataset))
#val_size = len(train_dataset) - train_size
#train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

test_transform = transforms.Compose([
    transforms.ToPILImage(),  # Convert numpy array to PIL Image
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))
])

batch_test_dict = load_cifar_batch(os.path.join(cifar10_dir, 'test_batch'))
val_images = batch_test_dict[b'data'].reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)
val_labels = np.array(batch_test_dict[b'labels'])

val_dataset = CustomCIFAR10Dataset(val_images, val_labels, transform=test_transform)

# DataLoaders
train_loader = DataLoader(train_dataset, batch_size=256, shuffle=True, num_workers=4)
val_loader = DataLoader(val_dataset, batch_size=256, shuffle=False, num_workers=4)

# Load test dataset
cifar_test_path = '/content/cifar_test_nolabel.pkl'
test_batch = load_cifar_batch(cifar_test_path)
test_images = test_batch[b'data'].astype(np.float32) / 255.0

# Convert test dataset to Tensor
test_dataset = [(test_transform(img),) for img in test_images]
test_loader = DataLoader(test_dataset, batch_size=256, shuffle=False, num_workers=4)

# Train function + plot
def train_model(model, train_loader, val_loader, epochs=50):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4)
    scheduler = MultiStepLR(optimizer, milestones=[30, 60, 80, 90], gamma=0.1)

    train_losses = []
    val_losses = []
    train_accuracies = []
    val_accuracies = []

    for epoch in range(epochs):
        # Training Phase
        model.train()
        running_loss = 0.0
        correct = 0
        total = 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()

            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        train_loss = running_loss / len(train_loader)
        train_acc = 100 * correct / total
        train_losses.append(train_loss)
        train_accuracies.append(train_acc)

        # Validation Phase
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for images, labels in val_loader:
```

```
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        val_loss /= len(val_loader)
        val_acc = 100 * correct / total
        val_losses.append(val_loss)
        val_accuracies.append(val_acc)

        scheduler.step()
        print(f'Epoch {epoch+1}, Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Val Loss: {val_

    # Plot Losses
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, epochs + 1), train_losses, label='Train Loss', color='red')
    plt.plot(range(1, epochs + 1), val_losses, label='Validation Loss', color='blue')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Train Loss & Validation Loss')
    plt.legend()
    plt.grid()
    plt.show()

    # Plot Accuracies
    plt.figure(figsize=(10, 5))
    plt.plot(range(1, epochs + 1), train_accuracies, label='Train Accuracy', color='green')
    plt.plot(range(1, epochs + 1), val_accuracies, label='Validation Accuracy', color='purple')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy (%)')
    plt.title('Train Accuracy & Validation Accuracy')
    plt.legend()
    plt.grid()
    plt.show()

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader
import torch.optim.lr_scheduler as lr_scheduler

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define the Squeeze-and-Excitation (SE) Block
class SEBlock(nn.Module):
    def __init__(self, channels, reduction=16):
        super(SEBlock, self).__init__()
        self.global_avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc1 = nn.Linear(channels, channels // reduction, bias=False)
        self.relu = nn.ReLU(inplace=True)
        self.fc2 = nn.Linear(channels // reduction, channels, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        b, c, _, _ = x.size()
        y = self.global_avg_pool(x).view(b, c)
        y = self.relu(self.fc1(y))
        y = self.sigmoid(self.fc2(y)).view(b, c, 1, 1)
        return x * y.expand_as(x)

# Define Residual Block with Shortcut Connections
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_sizes, stride=1, use_se=True):
        super(ResidualBlock, self).__init__()
        mid_channels = out_channels // 2

        self.conv1 = nn.Conv2d(in_channels, mid_channels, kernel_size=kernel_sizes[0], padding=1, bias=Fals
        self.bn1 = nn.BatchNorm2d(mid_channels)
```

```
        self.conv2 = nn.Conv2d(mid_channels, mid_channels, kernel_size=kernel_sizes[1], padding=1, stride=s
        self.bn2 = nn.BatchNorm2d(mid_channels)

        self.conv3 = nn.Conv2d(mid_channels, out_channels, kernel_size=kernel_sizes[2], padding=1, bias=Fa
        self.bn3 = nn.BatchNorm2d(out_channels)

        self.se = SEBlock(out_channels) if use_se else nn.Identity()
        self.leaky_relu = nn.LeakyReLU(0.1, inplace=True)
        self.dropout = nn.Dropout2d(0.2)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        identity = self.shortcut(x)
        out = self.leaky_relu(self.bn1(self.conv1(x)))
        out = self.leaky_relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out = self.se(out)
        out = self.dropout(out)
        out += identity
        return self.leaky_relu(out)

class CustomResNet_v4(nn.Module):
    def __init__(self, num_classes=10):
        super(CustomResNet_v4, self).__init__()
        # Initial convolution layer with a small increase in channels
        self.init_conv = nn.Conv2d(3, 96, kernel_size=3, stride=1, padding=1, bias=False)
        self.init_bn = nn.BatchNorm2d(96)
        self.leaky_relu = nn.LeakyReLU(0.1, inplace=True)

        # First residual block with slightly increased channels
        self.layer1 = self._make_layer(96, 124, [3, 3, 3], 4, stride=1)
        # Second residual block with slightly increased channels
        self.layer2 = self._make_layer(124, 189, [3, 3, 3], 4, stride=2)
        # Third residual block with slightly increased channels
        self.layer3 = self._make_layer(189, 280, [3, 3, 3], 3, stride=2)

        # Final average pooling and fully connected layers
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(280, num_classes)

    def _make_layer(self, in_channels, out_channels, kernel_sizes, blocks, stride):
        layers = [ResidualBlock(in_channels, out_channels, kernel_sizes, stride)]
        for _ in range(1, blocks):
            layers.append(ResidualBlock(out_channels, out_channels, kernel_sizes, stride=1))
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.leaky_relu(self.init_bn(self.init_conv(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.avg_pool(out)
        out = torch.flatten(out, 1)
        out = self.fc(out)
        return out



# Instantiate the model
model = CustomResNet_v4().to(device)

# Define the optimizer, scheduler, and loss function
optimizer = optim.SGD(model.parameters(), lr=0.05, momentum=0.9, weight_decay=0.0005)
scheduler = lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)
criterion = nn.CrossEntropyLoss()

# Print the model summary
from torchsummary import summary
```

```
summary(model, (3, 32, 32))

# Train the model
train_model(model, train_loader, val_loader, epochs=70) #change epoch

# Generate submission file
model.eval()
predictions = []
with torch.no_grad():
    for batch in test_loader:
        images = batch[0].to(device)  # Get images tensor from tuple and move to device
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.cpu().numpy())

# Generate submission file
submission = pd.DataFrame({'ID': np.arange(len(predictions)), 'Labels': predictions})
submission.to_csv('/content/submission1.csv', index=False)
print("Submission1 file saved.")
```

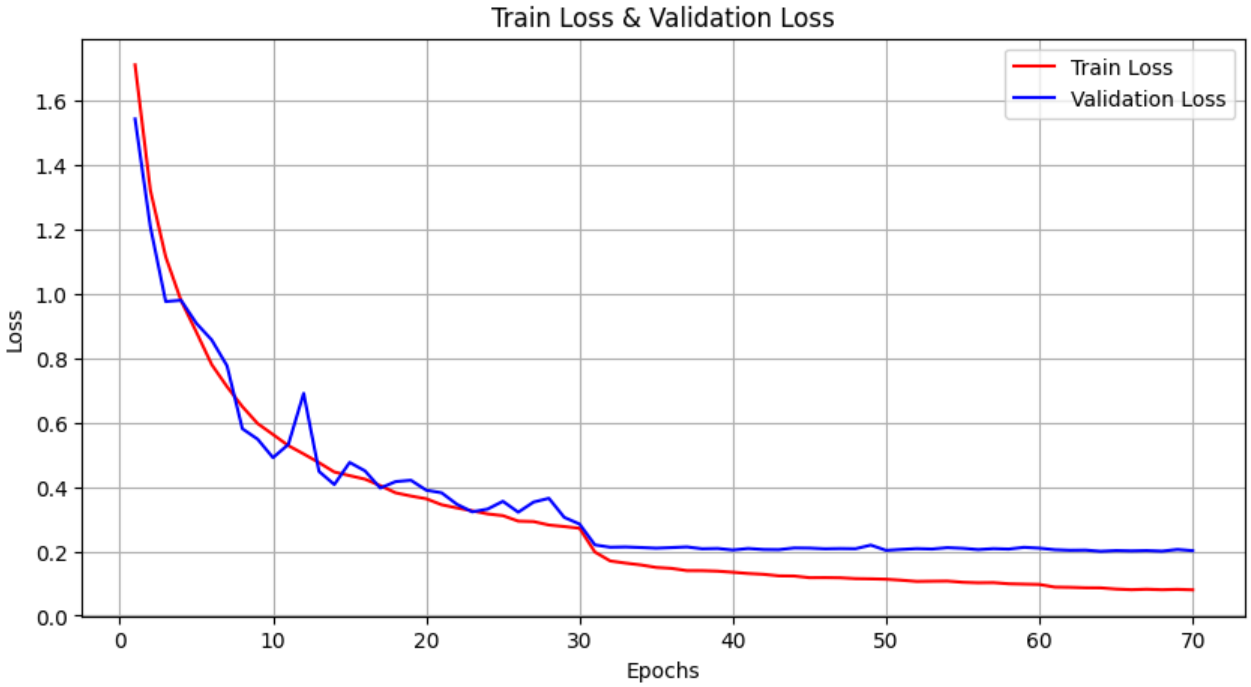
```
Using device: cuda
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624: UserWarning: This DataLoader wi
ll create 4 worker processes in total. Our suggested max number of worker in current system is 2, which is
smaller than what this DataLoader is going to create. Please be aware that excessive worker creation might
get DataLoader running slow or even freeze, lower the worker number to avoid potential slowness/freeze if n
ecessary.
  warnings.warn(
```

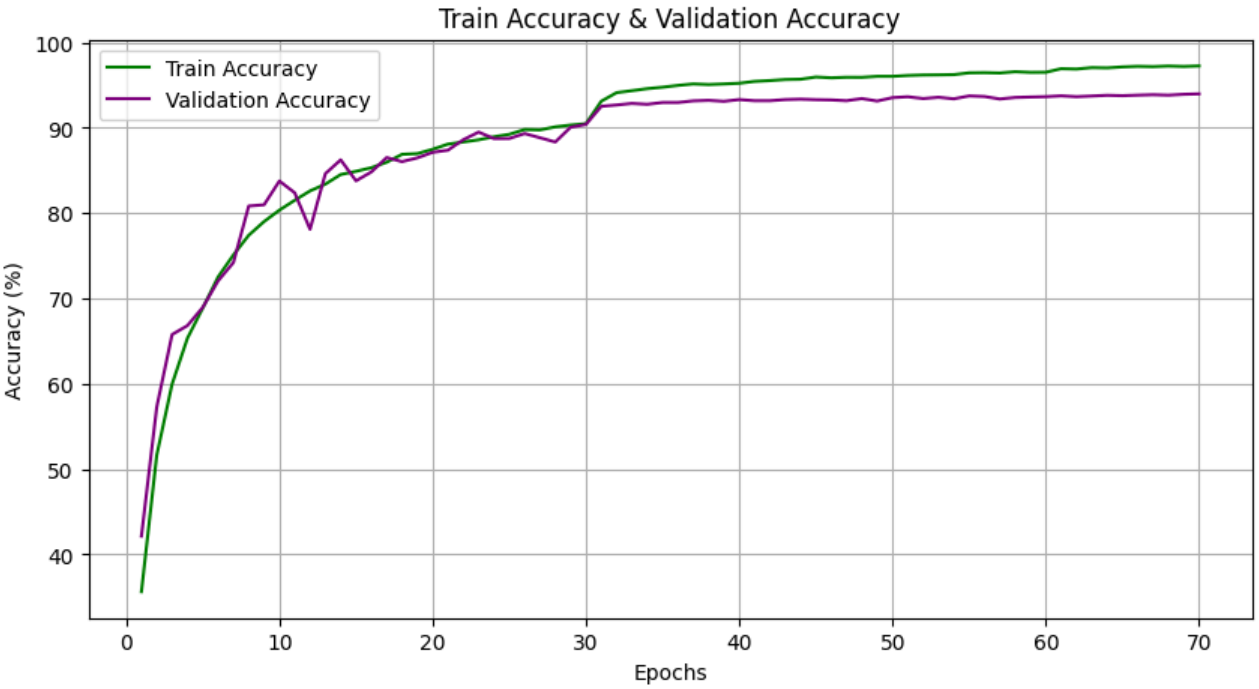
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 96, 32, 32]           2,592
       BatchNorm2d-2           [-1, 96, 32, 32]             192
         LeakyReLU-3           [-1, 96, 32, 32]               0
            Conv2d-4          [-1, 124, 32, 32]          11,904
       BatchNorm2d-5          [-1, 124, 32, 32]             248
            Conv2d-6           [-1, 62, 32, 32]          53,568
       BatchNorm2d-7           [-1, 62, 32, 32]             124
         LeakyReLU-8           [-1, 62, 32, 32]               0
            Conv2d-9           [-1, 62, 32, 32]          34,596
      BatchNorm2d-10           [-1, 62, 32, 32]             124
        LeakyReLU-11           [-1, 62, 32, 32]               0
           Conv2d-12          [-1, 124, 32, 32]          69,192
      BatchNorm2d-13          [-1, 124, 32, 32]             248
AdaptiveAvgPool2d-14            [-1, 124, 1, 1]               0
           Linear-15                   [-1, 7]             868
             ReLU-16                   [-1, 7]               0
           Linear-17                 [-1, 124]             868
          Sigmoid-18                 [-1, 124]               0
          SEBlock-19          [-1, 124, 32, 32]               0
        Dropout2d-20          [-1, 124, 32, 32]               0
        LeakyReLU-21          [-1, 124, 32, 32]               0
    ResidualBlock-22          [-1, 124, 32, 32]               0
           Conv2d-23           [-1, 62, 32, 32]          69,192
      BatchNorm2d-24           [-1, 62, 32, 32]             124
        LeakyReLU-25           [-1, 62, 32, 32]               0
           Conv2d-26           [-1, 62, 32, 32]          34,596
      BatchNorm2d-27           [-1, 62, 32, 32]             124
        LeakyReLU-28           [-1, 62, 32, 32]               0
           Conv2d-29          [-1, 124, 32, 32]          69,192
      BatchNorm2d-30          [-1, 124, 32, 32]             248
AdaptiveAvgPool2d-31            [-1, 124, 1, 1]               0
           Linear-32                   [-1, 7]             868
             ReLU-33                   [-1, 7]               0
           Linear-34                 [-1, 124]             868
          Sigmoid-35                 [-1, 124]               0
          SEBlock-36          [-1, 124, 32, 32]               0
        Dropout2d-37          [-1, 124, 32, 32]               0
        LeakyReLU-38          [-1, 124, 32, 32]               0
    ResidualBlock-39          [-1, 124, 32, 32]               0
           Conv2d-40           [-1, 62, 32, 32]          69,192
      BatchNorm2d-41           [-1, 62, 32, 32]             124
        LeakyReLU-42           [-1, 62, 32, 32]               0
           Conv2d-43           [-1, 62, 32, 32]          34,596
      BatchNorm2d-44           [-1, 62, 32, 32]             124
        LeakyReLU-45           [-1, 62, 32, 32]               0
           Conv2d-46          [-1, 124, 32, 32]          69,192
      BatchNorm2d-47          [-1, 124, 32, 32]             248
AdaptiveAvgPool2d-48            [-1, 124, 1, 1]               0
           Linear-49                   [-1, 7]             868
             ReLU-50                   [-1, 7]               0
           Linear-51                 [-1, 124]             868
          Sigmoid-52                 [-1, 124]               0
          SEBlock-53          [-1, 124, 32, 32]               0
        Dropout2d-54          [-1, 124, 32, 32]               0
        LeakyReLU-55          [-1, 124, 32, 32]               0
    ResidualBlock-56          [-1, 124, 32, 32]               0
           Conv2d-57           [-1, 62, 32, 32]          69,192
      BatchNorm2d-58           [-1, 62, 32, 32]             124
        LeakyReLU-59           [-1, 62, 32, 32]               0
           Conv2d-60           [-1, 62, 32, 32]          34,596
      BatchNorm2d-61           [-1, 62, 32, 32]             124
        LeakyReLU-62           [-1, 62, 32, 32]               0
           Conv2d-63          [-1, 124, 32, 32]          69,192
      BatchNorm2d-64          [-1, 124, 32, 32]             248
AdaptiveAvgPool2d-65            [-1, 124, 1, 1]               0
           Linear-66                   [-1, 7]             868
             ReLU-67                   [-1, 7]               0
           Linear-68                 [-1, 124]             868
          Sigmoid-69                 [-1, 124]               0
          SEBlock-70          [-1, 124, 32, 32]               0
        Dropout2d-71          [-1, 124, 32, 32]               0
        LeakyReLU-72          [-1, 124, 32, 32]               0
```

```
    ResidualBlock-73          [-1, 124, 32, 32]               0
          Conv2d-74          [-1, 189, 16, 16]          23,436
     BatchNorm2d-75          [-1, 189, 16, 16]             378
          Conv2d-76           [-1, 94, 32, 32]         104,904
     BatchNorm2d-77           [-1, 94, 32, 32]             188
       LeakyReLU-78           [-1, 94, 32, 32]               0
          Conv2d-79           [-1, 94, 16, 16]          79,524
     BatchNorm2d-80           [-1, 94, 16, 16]             188
       LeakyReLU-81           [-1, 94, 16, 16]               0
          Conv2d-82          [-1, 189, 16, 16]         159,894
     BatchNorm2d-83          [-1, 189, 16, 16]             378
AdaptiveAvgPool2d-84            [-1, 189, 1, 1]               0
          Linear-85                  [-1, 11]           2,079
            ReLU-86                  [-1, 11]               0
          Linear-87                 [-1, 189]           2,079
         Sigmoid-88                 [-1, 189]               0
         SEBlock-89          [-1, 189, 16, 16]               0
      Dropout2d-90          [-1, 189, 16, 16]               0
       LeakyReLU-91          [-1, 189, 16, 16]               0
    ResidualBlock-92          [-1, 189, 16, 16]               0
          Conv2d-93           [-1, 94, 16, 16]         159,894
     BatchNorm2d-94           [-1, 94, 16, 16]             188
       LeakyReLU-95           [-1, 94, 16, 16]               0
          Conv2d-96           [-1, 94, 16, 16]          79,524
     BatchNorm2d-97           [-1, 94, 16, 16]             188
       LeakyReLU-98           [-1, 94, 16, 16]               0
          Conv2d-99          [-1, 189, 16, 16]         159,894
    BatchNorm2d-100          [-1, 189, 16, 16]             378
AdaptiveAvgPool2d-101            [-1, 189, 1, 1]               0
         Linear-102                  [-1, 11]           2,079
           ReLU-103                  [-1, 11]               0
         Linear-104                 [-1, 189]           2,079
        Sigmoid-105                 [-1, 189]               0
        SEBlock-106          [-1, 189, 16, 16]               0
     Dropout2d-107          [-1, 189, 16, 16]               0
      LeakyReLU-108          [-1, 189, 16, 16]               0
   ResidualBlock-109          [-1, 189, 16, 16]               0
          Conv2d-110           [-1, 94, 16, 16]         159,894
     BatchNorm2d-111           [-1, 94, 16, 16]             188
       LeakyReLU-112           [-1, 94, 16, 16]               0
          Conv2d-113           [-1, 94, 16, 16]          79,524
     BatchNorm2d-114           [-1, 94, 16, 16]             188
       LeakyReLU-115           [-1, 94, 16, 16]               0
          Conv2d-116          [-1, 189, 16, 16]         159,894
     BatchNorm2d-117          [-1, 189, 16, 16]             378
AdaptiveAvgPool2d-118            [-1, 189, 1, 1]               0
         Linear-119                  [-1, 11]           2,079
           ReLU-120                  [-1, 11]               0
         Linear-121                 [-1, 189]           2,079
        Sigmoid-122                 [-1, 189]               0
        SEBlock-123          [-1, 189, 16, 16]               0
     Dropout2d-124          [-1, 189, 16, 16]               0
      LeakyReLU-125          [-1, 189, 16, 16]               0
   ResidualBlock-126          [-1, 189, 16, 16]               0
          Conv2d-127           [-1, 94, 16, 16]         159,894
     BatchNorm2d-128           [-1, 94, 16, 16]             188
       LeakyReLU-129           [-1, 94, 16, 16]               0
          Conv2d-130           [-1, 94, 16, 16]          79,524
     BatchNorm2d-131           [-1, 94, 16, 16]             188
       LeakyReLU-132           [-1, 94, 16, 16]               0
          Conv2d-133          [-1, 189, 16, 16]         159,894
     BatchNorm2d-134          [-1, 189, 16, 16]             378
AdaptiveAvgPool2d-135            [-1, 189, 1, 1]               0
         Linear-136                  [-1, 11]           2,079
           ReLU-137                  [-1, 11]               0
         Linear-138                 [-1, 189]           2,079
        Sigmoid-139                 [-1, 189]               0
        SEBlock-140          [-1, 189, 16, 16]               0
     Dropout2d-141          [-1, 189, 16, 16]               0
      LeakyReLU-142          [-1, 189, 16, 16]               0
   ResidualBlock-143          [-1, 189, 16, 16]               0
          Conv2d-144            [-1, 280, 8, 8]          52,920
     BatchNorm2d-145            [-1, 280, 8, 8]             560
          Conv2d-146          [-1, 140, 16, 16]         238,140
     BatchNorm2d-147          [-1, 140, 16, 16]             280
```

```
       LeakyReLU-148          [-1, 140, 16, 16]               0
          Conv2d-149            [-1, 140, 8, 8]         176,400
     BatchNorm2d-150            [-1, 140, 8, 8]             280
       LeakyReLU-151            [-1, 140, 8, 8]               0
          Conv2d-152            [-1, 280, 8, 8]         352,800
     BatchNorm2d-153            [-1, 280, 8, 8]             560
AdaptiveAvgPool2d-154            [-1, 280, 1, 1]               0
          Linear-155                  [-1, 17]           4,760
            ReLU-156                  [-1, 17]               0
          Linear-157                 [-1, 280]           4,760
         Sigmoid-158                 [-1, 280]               0
         SEBlock-159            [-1, 280, 8, 8]               0
       Dropout2d-160            [-1, 280, 8, 8]               0
       LeakyReLU-161            [-1, 280, 8, 8]               0
   ResidualBlock-162            [-1, 280, 8, 8]               0
          Conv2d-163            [-1, 140, 8, 8]         352,800
     BatchNorm2d-164            [-1, 140, 8, 8]             280
       LeakyReLU-165            [-1, 140, 8, 8]               0
          Conv2d-166            [-1, 140, 8, 8]         176,400
     BatchNorm2d-167            [-1, 140, 8, 8]             280
       LeakyReLU-168            [-1, 140, 8, 8]               0
          Conv2d-169            [-1, 280, 8, 8]         352,800
     BatchNorm2d-170            [-1, 280, 8, 8]             560
AdaptiveAvgPool2d-171            [-1, 280, 1, 1]               0
          Linear-172                  [-1, 17]           4,760
            ReLU-173                  [-1, 17]               0
          Linear-174                 [-1, 280]           4,760
         Sigmoid-175                 [-1, 280]               0
         SEBlock-176            [-1, 280, 8, 8]               0
       Dropout2d-177            [-1, 280, 8, 8]               0
       LeakyReLU-178            [-1, 280, 8, 8]               0
   ResidualBlock-179            [-1, 280, 8, 8]               0
          Conv2d-180            [-1, 140, 8, 8]         352,800
     BatchNorm2d-181            [-1, 140, 8, 8]             280
       LeakyReLU-182            [-1, 140, 8, 8]               0
          Conv2d-183            [-1, 140, 8, 8]         176,400
     BatchNorm2d-184            [-1, 140, 8, 8]             280
       LeakyReLU-185            [-1, 140, 8, 8]               0
          Conv2d-186            [-1, 280, 8, 8]         352,800
     BatchNorm2d-187            [-1, 280, 8, 8]             560
AdaptiveAvgPool2d-188            [-1, 280, 1, 1]               0
          Linear-189                  [-1, 17]           4,760
            ReLU-190                  [-1, 17]               0
          Linear-191                 [-1, 280]           4,760
         Sigmoid-192                 [-1, 280]               0
         SEBlock-193            [-1, 280, 8, 8]               0
       Dropout2d-194            [-1, 280, 8, 8]               0
       LeakyReLU-195            [-1, 280, 8, 8]               0
   ResidualBlock-196            [-1, 280, 8, 8]               0
AdaptiveAvgPool2d-197            [-1, 280, 1, 1]               0
          Linear-198                  [-1, 10]           2,810
================================================================
Total params: 4,905,430
Trainable params: 4,905,430
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 59.35
Params size (MB): 18.71
Estimated Total Size (MB): 78.08
----------------------------------------------------------------
Epoch 1, Train Loss: 1.7110, Train Acc: 35.65%, Val Loss: 1.5430, Val Acc: 42.16%
Epoch 2, Train Loss: 1.3212, Train Acc: 51.65%, Val Loss: 1.2077, Val Acc: 57.29%
Epoch 3, Train Loss: 1.1145, Train Acc: 59.97%, Val Loss: 0.9759, Val Acc: 65.78%
Epoch 4, Train Loss: 0.9789, Train Acc: 65.34%, Val Loss: 0.9797, Val Acc: 66.82%
Epoch 5, Train Loss: 0.8809, Train Acc: 68.88%, Val Loss: 0.9081, Val Acc: 68.96%
Epoch 6, Train Loss: 0.7801, Train Acc: 72.53%, Val Loss: 0.8567, Val Acc: 72.06%
Epoch 7, Train Loss: 0.7115, Train Acc: 75.11%, Val Loss: 0.7762, Val Acc: 74.21%
Epoch 8, Train Loss: 0.6495, Train Acc: 77.41%, Val Loss: 0.5807, Val Acc: 80.84%
Epoch 9, Train Loss: 0.5967, Train Acc: 79.02%, Val Loss: 0.5482, Val Acc: 80.98%
Epoch 10, Train Loss: 0.5628, Train Acc: 80.35%, Val Loss: 0.4909, Val Acc: 83.77%
Epoch 11, Train Loss: 0.5284, Train Acc: 81.51%, Val Loss: 0.5318, Val Acc: 82.39%
Epoch 12, Train Loss: 0.5024, Train Acc: 82.61%, Val Loss: 0.6906, Val Acc: 78.11%
Epoch 13, Train Loss: 0.4754, Train Acc: 83.40%, Val Loss: 0.4482, Val Acc: 84.63%
Epoch 14, Train Loss: 0.4463, Train Acc: 84.53%, Val Loss: 0.4074, Val Acc: 86.25%
```

```
Epoch 15, Train Loss: 0.4351, Train Acc: 84.91%, Val Loss: 0.4760, Val Acc: 83.79%
Epoch 16, Train Loss: 0.4241, Train Acc: 85.33%, Val Loss: 0.4498, Val Acc: 84.83%
Epoch 17, Train Loss: 0.4039, Train Acc: 85.97%, Val Loss: 0.3965, Val Acc: 86.52%
Epoch 18, Train Loss: 0.3816, Train Acc: 86.90%, Val Loss: 0.4165, Val Acc: 86.04%
Epoch 19, Train Loss: 0.3717, Train Acc: 86.98%, Val Loss: 0.4206, Val Acc: 86.48%
Epoch 20, Train Loss: 0.3632, Train Acc: 87.49%, Val Loss: 0.3898, Val Acc: 87.13%
Epoch 21, Train Loss: 0.3446, Train Acc: 88.10%, Val Loss: 0.3817, Val Acc: 87.38%
Epoch 22, Train Loss: 0.3346, Train Acc: 88.34%, Val Loss: 0.3460, Val Acc: 88.61%
Epoch 23, Train Loss: 0.3257, Train Acc: 88.60%, Val Loss: 0.3226, Val Acc: 89.50%
Epoch 24, Train Loss: 0.3162, Train Acc: 88.95%, Val Loss: 0.3312, Val Acc: 88.74%
Epoch 25, Train Loss: 0.3107, Train Acc: 89.24%, Val Loss: 0.3553, Val Acc: 88.75%
Epoch 26, Train Loss: 0.2937, Train Acc: 89.80%, Val Loss: 0.3214, Val Acc: 89.33%
Epoch 27, Train Loss: 0.2924, Train Acc: 89.76%, Val Loss: 0.3529, Val Acc: 88.84%
Epoch 28, Train Loss: 0.2816, Train Acc: 90.11%, Val Loss: 0.3646, Val Acc: 88.34%
Epoch 29, Train Loss: 0.2769, Train Acc: 90.33%, Val Loss: 0.3054, Val Acc: 90.07%
Epoch 30, Train Loss: 0.2715, Train Acc: 90.49%, Val Loss: 0.2846, Val Acc: 90.40%
Epoch 31, Train Loss: 0.1977, Train Acc: 93.12%, Val Loss: 0.2199, Val Acc: 92.52%
Epoch 32, Train Loss: 0.1700, Train Acc: 94.12%, Val Loss: 0.2126, Val Acc: 92.68%
Epoch 33, Train Loss: 0.1634, Train Acc: 94.35%, Val Loss: 0.2135, Val Acc: 92.86%
Epoch 34, Train Loss: 0.1576, Train Acc: 94.59%, Val Loss: 0.2118, Val Acc: 92.76%
Epoch 35, Train Loss: 0.1499, Train Acc: 94.76%, Val Loss: 0.2098, Val Acc: 92.97%
Epoch 36, Train Loss: 0.1470, Train Acc: 94.97%, Val Loss: 0.2116, Val Acc: 92.97%
Epoch 37, Train Loss: 0.1400, Train Acc: 95.14%, Val Loss: 0.2136, Val Acc: 93.16%
Epoch 38, Train Loss: 0.1399, Train Acc: 95.07%, Val Loss: 0.2078, Val Acc: 93.22%
Epoch 39, Train Loss: 0.1383, Train Acc: 95.14%, Val Loss: 0.2088, Val Acc: 93.11%
Epoch 40, Train Loss: 0.1349, Train Acc: 95.22%, Val Loss: 0.2041, Val Acc: 93.31%
Epoch 41, Train Loss: 0.1311, Train Acc: 95.44%, Val Loss: 0.2088, Val Acc: 93.18%
Epoch 42, Train Loss: 0.1286, Train Acc: 95.54%, Val Loss: 0.2056, Val Acc: 93.18%
Epoch 43, Train Loss: 0.1238, Train Acc: 95.66%, Val Loss: 0.2052, Val Acc: 93.31%
Epoch 44, Train Loss: 0.1232, Train Acc: 95.69%, Val Loss: 0.2104, Val Acc: 93.36%
Epoch 45, Train Loss: 0.1181, Train Acc: 95.96%, Val Loss: 0.2100, Val Acc: 93.30%
Epoch 46, Train Loss: 0.1183, Train Acc: 95.86%, Val Loss: 0.2078, Val Acc: 93.27%
Epoch 47, Train Loss: 0.1178, Train Acc: 95.93%, Val Loss: 0.2085, Val Acc: 93.18%
Epoch 48, Train Loss: 0.1151, Train Acc: 95.92%, Val Loss: 0.2080, Val Acc: 93.43%
Epoch 49, Train Loss: 0.1144, Train Acc: 96.03%, Val Loss: 0.2192, Val Acc: 93.14%
Epoch 50, Train Loss: 0.1133, Train Acc: 96.03%, Val Loss: 0.2029, Val Acc: 93.54%
Epoch 51, Train Loss: 0.1102, Train Acc: 96.14%, Val Loss: 0.2059, Val Acc: 93.64%
Epoch 52, Train Loss: 0.1065, Train Acc: 96.19%, Val Loss: 0.2083, Val Acc: 93.43%
Epoch 53, Train Loss: 0.1071, Train Acc: 96.21%, Val Loss: 0.2071, Val Acc: 93.58%
Epoch 54, Train Loss: 0.1075, Train Acc: 96.24%, Val Loss: 0.2114, Val Acc: 93.40%
Epoch 55, Train Loss: 0.1038, Train Acc: 96.44%, Val Loss: 0.2093, Val Acc: 93.74%
Epoch 56, Train Loss: 0.1023, Train Acc: 96.47%, Val Loss: 0.2054, Val Acc: 93.67%
Epoch 57, Train Loss: 0.1027, Train Acc: 96.43%, Val Loss: 0.2083, Val Acc: 93.38%
Epoch 58, Train Loss: 0.0991, Train Acc: 96.58%, Val Loss: 0.2069, Val Acc: 93.56%
Epoch 59, Train Loss: 0.0980, Train Acc: 96.50%, Val Loss: 0.2124, Val Acc: 93.61%
Epoch 60, Train Loss: 0.0970, Train Acc: 96.51%, Val Loss: 0.2098, Val Acc: 93.65%
Epoch 61, Train Loss: 0.0888, Train Acc: 96.93%, Val Loss: 0.2050, Val Acc: 93.75%
Epoch 62, Train Loss: 0.0881, Train Acc: 96.89%, Val Loss: 0.2032, Val Acc: 93.65%
Epoch 63, Train Loss: 0.0867, Train Acc: 97.06%, Val Loss: 0.2037, Val Acc: 93.73%
Epoch 64, Train Loss: 0.0865, Train Acc: 97.03%, Val Loss: 0.2000, Val Acc: 93.81%
Epoch 65, Train Loss: 0.0827, Train Acc: 97.14%, Val Loss: 0.2023, Val Acc: 93.77%
Epoch 66, Train Loss: 0.0808, Train Acc: 97.21%, Val Loss: 0.2013, Val Acc: 93.83%
Epoch 67, Train Loss: 0.0821, Train Acc: 97.17%, Val Loss: 0.2023, Val Acc: 93.88%
Epoch 68, Train Loss: 0.0807, Train Acc: 97.24%, Val Loss: 0.2005, Val Acc: 93.83%
Epoch 69, Train Loss: 0.0818, Train Acc: 97.19%, Val Loss: 0.2057, Val Acc: 93.93%
Epoch 70, Train Loss: 0.0803, Train Acc: 97.25%, Val Loss: 0.2019, Val Acc: 93.98%
```






```
Submission1 file saved.
```

In [ ]:
```
torch.save(model.state_dict(), '/content/model_checkpoint.pth')
```

In [ ]:
```
# Generate submission file
model.eval()
predictions = []
with torch.no_grad():
    for batch in test_loader:
        images = batch[0].to(device)  # Get images tensor from tuple and move to device
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        predictions.extend(predicted.cpu().numpy())

# Generate submission file
submission = pd.DataFrame({'ID': np.arange(len(predictions)), 'Labels': predictions})
submission.to_csv('/content/submission2.csv', index=False)
print("Submission1 file saved.")
```

```
Submission1 file saved.
```

In [ ]: