Assignment 1: Image Classification using Convolutional Neural Networks

Submitted by: Lakshya Prakash Agarwal (261149449)

Introduction

In this report, we aim to develop a Convolutional Neural Network (CNN) using the ResNet architecture to classify images from the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 different classes, with 50,000 training images and 10,000 test images. The primary goal is to preprocess the data, implement the ResNet architecture, train the model, and evaluate its performance using various metrics.

Load and Preprocess Data

Loading the Dataset

The CIFAR-10 dataset is loaded using the `torchvision.datasets` module. The dataset is split into training, validation, and test sets. The training set is further split to create a validation set without data leakage.

```
from torchvision.datasets import CIFAR10

# Load the CIFAR-10 dataset
train_dataset = CIFAR10(DATA_DIR, train=True, download=True)
val_dataset = CIFAR10(DATA_DIR, train=True, download=True)
test_dataset = CIFAR10(DATA_DIR, train=False, download=True)
```

Data Preprocessing

Data preprocessing involves normalizing the images and applying data augmentation techniques such as random horizontal flipping and random cropping. The mean and standard deviation of the dataset are calculated for normalization.

```
from torchvision import transforms

# Calculate mean and standard deviation
DATA_MEANS = (train_dataset.data / 255.0).mean(axis=(0, 1, 2))
DATA_STD = (train_dataset.data / 255.0).std(axis=(0, 1, 2))

# Define transformations
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomResizedCrop((32, 32), scale=(0.8, 1.0), ratio=(0.9, 1.1)),
    transforms.ToTensor(),
    transforms.Normalize(DATA_MEANS, DATA_STD),
])

test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(DATA_MEANS, DATA_STD),
])
```

Data Loaders

Data loaders are created for the training, validation, and test sets to facilitate batch processing.

```
from torch.utils.data import DataLoader

BATCH_SIZE = 128

# Create data loaders
train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=4, pin_memory=True, drop
...
```

ResNet: CNN Architecture

The ResNet architecture is implemented using PyTorch. ResNet is known for its residual learning framework, which helps in training very deep networks by addressing the vanishing gradient problem.

The convolutional neural network architecture used in this notebook is inspired by the ResNet architecture from the paper "Deep Residual Learning for Image Recognition" by He et al. (2015).

The ResNet architecture consists of a series of residual blocks, which are composed of convolutional layers, batch normalization, and ReLU activation functions. The residual blocks are connected by skip connections, which allow the network to learn residual functions instead of the original mapping. This helps to mitigate the vanishing gradient problem and enables the training of very deep networks. At the end of the network, a global average pooling layer and a fully connected layer are used to produce the final output that represents the class probabilities.

In essence, instead of learning the desired output, the network learns the residual between the desired output and the current output.

Architecture

The ResNet architecture used in this notebook consists of the following components:

- Initial convolutional layer with 16 output channels and kernel size 1x1, followed by batch normalization
- A series of residual blocks with feature map sizes of {32, 16, 8}
- A final output layer that consists of global average pooling and a fully connected layer with 10 output units (one for each class)

Residual block

The residual block consists of the following components:

- Convolutional layer with kernel size 3x3 and padding 1, with an optional stride of 2 for downsampling
- Batch normalization
- ReLU activation function
- Convolutional layer with kernel size 3x3 and padding 1 and a stride of 1
- Batch normalization
- Skip connection to add the input to the output of the second convolutional layer

The ResNet model is implemented using PyTorch and is defined in the `ResNet` class.

Training and Validation

To train the ResNet model, PyTorch Lightning is used to simplify the training process. PyTorch Lightning provides a high-level interface for PyTorch that abstracts away the training loop, validation loop, and other boilerplate code. This makes it easier to train models and experiment with different architectures and hyperparameters.

The training process consists of the following steps:

- Initialize the ResNet model with the specified hyperparameters (e.g., number of residual blocks, feature map sizes, etc.)
- Define the loss function (cross-entropy loss)
- Define the optimizer, associated hyperparameters (e.g., learning rate, weight decay) and learning rate scheduler
- Log the training, validation, and test accuracy using TensorBoard
- Log the validation images and predictions using TensorBoard for visualization
- Save the best model based on the validation accuracy
- Monitor the learning rate and log it using TensorBoard

The model is trained with 32-layers, a batch size of 128, a learning rate of 0.1, a weight decay of 1e-4, a momentum of 0.9 and a learning rate scheduler that reduces the learning rate on a plateau. The model is trained for 50 epochs, and the best model based on the validation accuracy is saved. The hyperparameters are chosen based on the original paper's recommendations for training ResNet on CIFAR-10.

Evaluation

The ResNet-32 model is evaluated on the validation and test sets to measure its performance. The accuracy on the test set is reported as the final evaluation metric.

Results

The results of the model evaluation are summarized in the following table:

Dataset	Accuracy
Validation	0.8716
Test	0.8677

Conclusion

In this report, we developed a Convolutional Neural Network using the ResNet architecture to classify images from the CIFAR-10 dataset. The model achieved an accuracy of 87.16% on the validation set and 86.77% on the test set. The ResNet architecture, with its residual learning framework, helped in training a deep network and achieving good performance on the image classification task. The model can be further improved by adding more layers, and using data augmentation techniques.