模式识别与机器学习大作业
PRML

王淯 尹超 伍昱衡 陈志强 崔祯徐 郑子辰
(按照姓氏笔画排序)
中国科学院大学，北京 10004
University of Chinese Academy of Sciences, Beijing 100049, China

2025.6.3 - 2025.6.30

序言

本文为笔者模式识别与机器学习的大作业。
望老师批评指正。

分工如下表所示：

姓名	分工
王滑	boosting 部分文章撰写
尹超	神经网络部分算法实现和文章撰写
伍昱衡	决策树部分文章撰写
陈志强	神经网络部分文章撰写
崔祯徐	boosting 部分算法实现
郑子辰	决策树部分算法实现

目录

Chapter 1 神经网络部分

1.1 神经网络简介

1.1.1 关键要点

- 神经网络是一种模拟人类大脑的计算模型，用于模式识别和预测。
- 不同结构如 CNN、RNN、Transformer 各有专长，适合不同任务。
- 研究表明，选择结构取决于数据类型和任务复杂性。

1.1.2 神经网络简介

神经网络（Neural Networks）是一种受生物神经系统启发的机器学习模型，广泛用于分类、回归和生成任务。它由多个节点（神经元）组成，这些节点通过加权连接传递信息，通过训练调整权重以学习数据模式。训练过程包括前向传播、损失计算和反向传播。

1.1.3 不同结构概述

神经网络的结构多样化，每种结构针对特定问题设计。以下是主要类型及其适用场景：

- **前向神经网络（FNN）**：适合静态数据，如基本分类。
- **卷积神经网络（CNN）**：专为图像处理设计，擅长提取空间特征。
- **循环神经网络（RNN）和 LSTM**：处理序列数据，如语言和时间序列。
- **Transformer**：用于自然语言处理，处理长距离依赖。
- **生成对抗网络（GAN）**：生成新数据，如图像生成。

1.1.4 结构差异

不同结构在数据处理方式和复杂性上存在显著差异。例如，CNN 通过卷积层提取图像特征，而 RNN 通过循环捕捉时间依赖。Transformer 则依赖注意力机制，适合并行计算。

1.1.5 详细调研笔记

神经网络的定义与基本原理

神经网络是一种计算模型，模仿人类大脑神经系统的结构和功能，由多个层组成，包括输入层、隐藏层和输出层。每个神经元通过加权连接接收输入，应用激活函数（如 ReLU、Sigmoid）引入非线性，并传递信号。训练过程通过前向传播计算输出，反向传播调整权重以最小化损失函数（如均方误差或交叉熵）。

根据 GeeksforGeeks 的《Neural Networks: A Beginner's Guide》（https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/），神经网络的学习过程包括输入计算、输出生成和参数迭代优化，广泛应用于模式识别和复杂问题解决。

不同神经网络结构的分类与差异

神经网络的结构多样化，以下是主要类型及其特点，基于 V7Labs 的《The Essential Guide to Neural Network Architectures》（https://www.v7labs.com/blog/neural-network-architectures-guide/）和 Wikipedia 的《Neural Network (Machine Learning)》（https://en.wikipedia.org/wiki/Neural_network_(machine_learning)）的综合分析：

table 1.1: 神经网络结构对比

结构	描述	关键特点	局限性	适用场景
FNN	数据单向流动，无循环	无反馈机制，适合静态数据	无法处理序列数据	基本分类、回归
MLP	FNN 扩展，含隐藏层	处理非线性，学习复杂特征	计算量较大	图像分类、语音识别
CNN	使用卷积和池化层	参数共享，提取空间特征	池化丢失空间关系	图像分析、物体检测、NLP
RNN	处理序列，循环连接	记忆功能，捕捉时间依赖	梯度消失，训练慢	NLP、时间序列预测
LSTM	RNN 增强，记忆单元	解决长序列梯度消失	训练速度慢	语音识别、机器翻译
GAN	生成器与判别器对抗	生成新数据，如图像、文本	训练不稳定	图像生成、数据增强
Transformer	基于注意力机制	处理长距离依赖，适合并行计算	计算复杂度高	NLP、机器翻译
ResNet	深层网络，跳跃连接	解决梯度消失，深层训练	高计算资源	图像分类、目标检测
Hopfield 网络	基于 Hebbian 学习	能量函数驱动，模式检索	不适合训练	模式识别、记忆任务
Boltzmann 机	无监督，生成式模型	随机能量函数，生成任务	训练复杂	深度生成模型
RBF 网络	功能近似，2013 年引入	最佳近似，非线性识别	结构与 MLP 不同	分类、非线性系统
Highway 网络	2015 年，开放门控	训练超深网络，解决退化	与 ResNet 类似	深层网络训练
Capsule 网络	改进 CNN，保留层次	本地胶囊，旋转鲁棒性	实现复杂	空间关系处理
MobileNet	轻量级，适合移动设备	深度可分离卷积	性能受限	移动设备、机器人

结构之间的关键差异

- **数据类型**: CNN 适合空间数据(如图像), RNN/LSTM 适合序列数据(如文本、时间序列), Transformer 适合长文本，GAN 专注于生成数据。
- **处理方式**：FNN 和 MLP 是静态的，RNN/LSTM 有记忆，Transformer 使用自注意力机制，CNN 通过卷积提取特征。
- **复杂性**：FNN 简单，ResNet 和 Transformer 更复杂，适合更深的网络和复杂任务。
- **训练难度**：RNN 存在梯度消失，LSTM 和 ResNet 通过设计解决此问题，Transformer 依赖大规模数据和计算资源。

根据 MyGreatLearning 的《Types of Neural Networks and Definition of Neural Network》(https://www.mygreatlearning.com/blog/types-of-neural-networks/)，不同结构的生物启发设计（如 ANN 模仿神经元）决定了其在复杂应用中的表现。

应用场景与局限性

- **CNN**：如 V7Labs 的《Convolutional Neural Networks Guide》（链接）所示，广泛用于图像分类和物体检测，但池化可能丢失空间信息。
- **RNN 和 LSTM**：如 V7Labs 的《Recurrent Neural Networks Guide》（链接）所述，适合 NLP 和时间序列，但训练慢，LSTM 缓解了长序列问题。
- **Transformer**：如《Attention Is All You Need》（链接）所示，主导 NLP 领域，但计算成本高。
- **GAN**：如《Generative Adversarial Networks》（链接）所述，生成高质量图像，但训练不稳定。

综合分析

神经网络的多样性使其能够适应各种任务，从简单的 FNN 到复杂的 Transformer，每种结构都有其独特优势。选择合适结构需考虑数据类型、任务复杂性和计算资源。根据 UpGrad 的《Neural Network Architecture: Types, Components & Key Algorithms》（链接），未来的研究可能进一步优化轻量级网络（如 MobileNet）以适应移动设备。

关键引用

- GeeksforGeeks Neural Networks Beginner's Guide: https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/
- V7Labs Essential Guide to Neural Network Architectures: https://www.v7labs.com/blog/neural-network-architectures-guide/
- Wikipedia Neural Network Machine Learning: https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
- MyGreatLearning Types of Neural Networks Definition: https://www.mygreatlearning.com/blog/types-of-neural-networks/
- UpGrad Neural Network Architecture Components Algorithms: https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/
- V7Labs Convolutional Neural Networks Guide: https://www.v7labs.com/blog/convolutional-neural-networks-guide/
- V7Labs Recurrent Neural Networks Guide: https://www.v7labs.com/blog/recurrent-neural-networks-guide/
- Attention Is All You Need Transformer Paper: https://arxiv.org/abs/1706.03762
- Generative Adversarial Networks NIPS Paper: https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

1.1.6 ResNet 与 Vision Transformer (ViT) 的结构对比

table 1.2: ResNet 与 Vision Transformer (ViT) 的详细结构对比

比较维度	ResNet	Vision Transformer (ViT)
架构类型	卷积神经网络（CNN）	Transformer 架构
提出年份	2015	2020
提出机构	微软研究院	Google Brain
基本单元	卷积层 + 残差连接（Residual Block）	自注意力模块（Multi-head Attention）+ MLP
参数量（Base 模型）	较少（如 ResNet-50 约 25M）	较多（如 ViT-B 约 86M）
计算复杂度	较低，主要是卷积操作	高，自注意力为 $O(n^2)$ 时间复杂度
输入处理	原始图像直接进入卷积网络	图像切成 Patch，再投影为序列
位置建模方式	隐式建模（卷积天然包含位置信息）	显式位置编码（Positional Encoding）
空间建模能力	局部为主，靠堆叠层数扩大全局感受野	全局建模能力强（自注意力机制）
可解释性	较强，可通过卷积特征图分析	较弱，注意力机制不易解释
收敛速度	快速，适合从头训练	慢，对初始化敏感
是否需要预训练	可以从头训练，也支持预训练	强烈依赖预训练（无预训练效果差）
数据规模依赖	中小规模数据也能表现良好	需要大规模数据（如 ImageNet-21k）
训练资源需求	普通 GPU 即可训练（如单卡）	需多卡/TPU，大内存显卡更佳
推理速度	快（卷积并行度高）	慢（序列操作限制并行度）
适合任务	图像分类、目标检测、语义分割等经典视觉任务	大规模视觉任务、跨模态学习、多任务联合建模
代表模型	ResNet-18/34/50/101/152	ViT-B/16, ViT-L/32, DeiT, Swin Transformer

1.2 CNN Training on CIFAR-10 Dataset

1.2.1 CNN training on CIFAR-10 dataset

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, SubsetRandomSampler
import numpy as np

# 数据预处理 - 增强数据增强
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616))
])

# 加载 CIFAR-10 数据集
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True,
    transform=transform_train)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True,
    transform=transform_test)

# 划分训练集和验证集
validation_split = 0.2  # 20% 用于验证集
dataset_size = len(trainset)
indices = list(range(dataset_size))
np.random.seed(42)  # 固定随机种子以确保可重复性
np.random.shuffle(indices)
split = int(np.floor(validation_split * dataset_size))
train_indices, val_indices = indices[split:], indices[:split]

# 创建 DataLoader
train_sampler = SubsetRandomSampler(train_indices)
val_sampler = SubsetRandomSampler(val_indices)

trainloader = DataLoader(trainset, batch_size=128, sampler=train_sampler, num_workers
    =2)
valloader = DataLoader(trainset, batch_size=128, sampler=val_sampler, num_workers=2)
testloader = DataLoader(testset, batch_size=128, shuffle=False, num_workers=2)

```

```python
# 定义改进的 CNN 模型
class ImprovedCNN(nn.Module):
    def __init__(self):
        super(ImprovedCNN, self).__init__()

        # 第一个卷积块
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2)
        )

        # 第二个卷积块
        self.conv2 = nn.Sequential(
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2)
        )

        # 第三个卷积块
        self.conv3 = nn.Sequential(
            nn.Conv2d(128, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2)
        )

        # 全连接层
        self.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 4 * 4, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, 10)
        )

    def forward(self, x):
```

```
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

# 主程序
if __name__ == '__main__':
    # 实例化模型
    model = ImprovedCNN()

    # 定义损失函数和优化器
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-4)
    # 学习率调度器
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', factor=0.1,
    patience=5, verbose=True)

    # 如果有 GPU，将模型移动到 GPU
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model.to(device)
    print(f"使用设备: {device}")

    # 训练模型
    best_val_acc = 0.0
    for epoch in range(100):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for i, (inputs, labels) in enumerate(trainloader):
            inputs, labels = inputs.to(device), labels.to(device)

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

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

            if i % 100 == 99:
                print(f'[{epoch + 1}, {i + 1}] loss: {running_loss / 100:.3f} | acc:
    {100.*correct/total:.2f}%')
```

```
                running_loss = 0.0

        # 每个epoch结束后验证
        model.eval()
        val_loss = 0
        correct = 0
        total = 0
        with torch.no_grad():
            for data in valloader:
                images, labels = data
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        val_acc = 100. * correct / total
        print(f'Epoch {epoch+1}: 验证准确率: {val_acc:.2f}%')

        # 更新学习率
        scheduler.step(val_loss)

        # 保存验证集上最佳模型
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            torch.save(model.state_dict(), 'best_cifar10_model.pth')
            print(f'保存最佳模型，验证准确率: {best_val_acc:.2f}%')

    print('训练完成')

    # 加载最佳模型进行测试
    model.load_state_dict(torch.load('best_cifar10_model.pth'))
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'最佳模型在测试集上的准确率: {100 * correct / total:.2f}%')
```

Listing 1.1: 神经网络 CNN 训练（纯手写）

1.2.2 CNN 训练结果

figure 1.1: CNN 训练预测示例

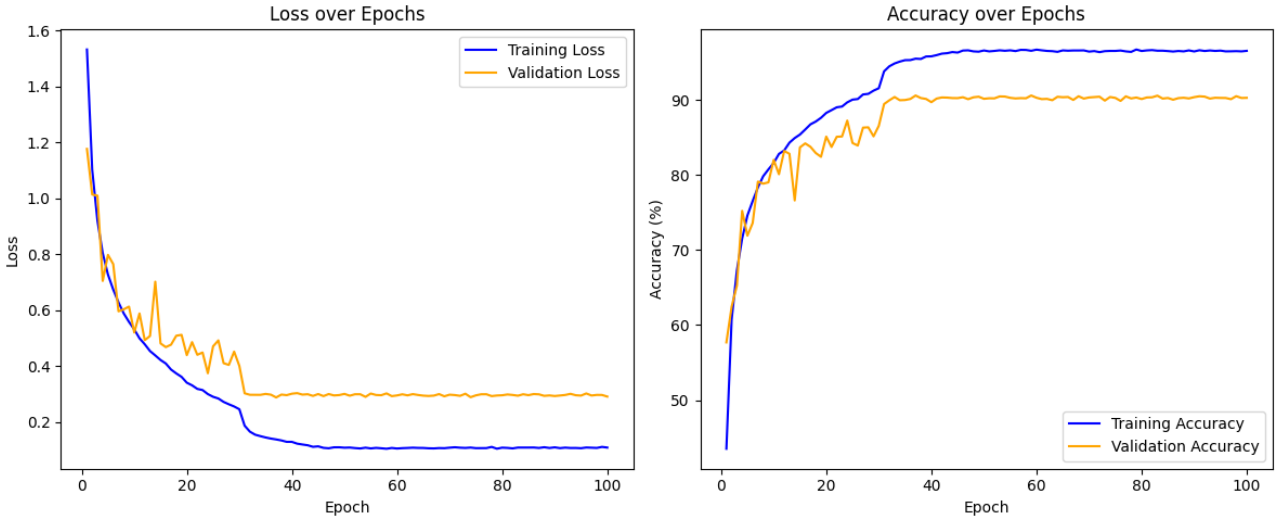

figure 1.2: CNN's Loss and Accuracy on Training and Validation Sets

1.3 Convolutional Neural Networks for CIFAR-10

1.3.1 Overview of CNNs

Convolutional Neural Networks (CNNs) are specialized architectures tailored for processing grid-like data, such as images, by leveraging spatial hierarchies. Their efficacy in image recognition, particularly for datasets like CIFAR-10, stems from their ability to extract local features while maintaining computational efficiency [1].

Limitations of Traditional Neural Networks

Traditional neural networks struggle with image data due to:

i. *High Input Dimensionality*: A 1000×1000 pixel color image requires 3,000,000 input neurons (1000 × 1000 × 3 for RGB channels), posing significant computational challenges.

ii. *Parameter Overload*: Connecting these to a 1000-neuron hidden layer demands 3,000,000,000 weights, complicating training and storage.

iii. *Loss of Spatial Context*: Flattening images into vectors discards pixel relationships critical for recognizing patterns like edges or textures.

iv. *Lack of Translation Invariance*: Object shifts in images can disrupt recognition, as these networks rely on fixed pixel patterns.

Advantages of CNNs

CNNs overcome these issues through:

i. *Local Receptive Fields*: Neurons connect to small input regions (e.g., 3×3 patches), reducing parameters.

ii. *Parameter Sharing*: Filters with fixed weights slide across the image, minimizing parameters and ensuring features like edges are detected universally.

iii. *Spatial Downsampling*: Pooling layers (e.g., max pooling) summarize regions, lowering dimensions and enhancing robustness to minor translations.

1.3.2 CNN Architecture

A typical CNN stacks convolutional blocks to extract hierarchical features, structured as follows:

i. *Input Layer*: Receives raw image data (e.g., 32×32×3 for CIFAR-10).

ii. *Convolutional Layer*: Applies filters to produce feature maps via:

$$S(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n),$$

where I is the input, K is the filter, and S is the feature map.

iii. *Activation Layer*: Introduces non-linearity with ReLU, $f(x) = \max(0, x)$, enabling complex pattern learning [2].

iv. *Pooling Layer*: Reduces dimensions, e.g., max pooling selects the maximum value in a 2×2 region.

v. *Stacked Blocks*: Early layers detect edges, intermediate layers capture textures, and deeper layers identify objects.

vi. *Flatten Layer*: Converts feature maps into a vector.

vii. *Fully Connected Layers*: Combine features for classification.

viii. *Output Layer*: Uses Softmax to produce class probabilities.

1.3.3 Implementation on CIFAR-10

The CIFAR-10 dataset, comprising 50,000 training and 10,000 test images across 10 classes, serves as a robust testbed [1].

Data Preprocessing

To enhance generalization:

- *Augmentation*: Random cropping and horizontal flipping diversify training data.
- *Normalization*: Images are standardized using CIFAR-10's mean and standard deviation.
- *Dataset Split*: 80% training, 20% validation, sampled via `SubsetRandomSampler`.
- *Batch Size*: Set to 128 for efficient training.

Model Design

The CNN features three convolutional blocks:

- *Structure*: Each block has two 3×3 convolutional layers, batch normalization [3], ReLU, and max pooling.
- *Channels*: Increase from 64 to 128 to 256, capturing complex features.
- *Spatial Reduction*: Image size reduces from 32×32 to 4×4.
- *Output*: A 4×4×256 feature map flattens to 4096 dimensions, followed by fully connected layers with 0.5 dropout [4].

table 1.3: CNN Architecture

Layer	Output Shape	Parameters	Operation
Conv Block 1	32×32×64	3×3 filters, BN, ReLU	Convolution + Pooling
Conv Block 2	16×16×128	3×3 filters, BN, ReLU	Convolution + Pooling
Conv Block 3	8×8×256	3×3 filters, BN, ReLU	Convolution + Pooling
Flatten	4096	–	Reshape
Fully Connected	10	Dropout (0.5)	Classification

Training Strategy

Training over 100 epochs utilized:

- *Optimizer*: SGD with momentum for faster convergence.
- *Regularization*: Weight decay (5×10^{-4}) to curb overfitting.
- *Scheduler*: Reduces learning rate by 10 if validation loss stalls for 5 epochs.

Training Workflow

Each epoch involved:

- *Monitoring*: Logging loss and accuracy every 100 batches.
- *Validation*: Evaluating performance post-epoch to adjust the learning rate.
- *Model Saving*: Retaining the best-performing model based on validation accuracy.

Evaluation

The model achieved ∼90% test accuracy, with correct classifications of diverse images (e.g., distinguishing cats from dogs).

1.3.4 Performance Analysis

By the 40th epoch, training accuracy exceeded 90% with losses below 10%, while validation accuracy reached ∼90% with losses around 30%. These metrics indicate robust learning, though validation performance suggests room for improvement.

1.3.5 Discussion

The 90% accuracy reflects effective use of CNNs, augmented by data preprocessing and regularization. Compared to state-of-the-art models like ResNet, which achieve over 95% [5], this model is solid but could benefit from deeper architectures or advanced techniques like residual connections. Early stopping around the 40th epoch could optimize training efficiency.

This implementation underscores CNNs'power for image classification and provides a foundation for further exploration in computer vision tasks.

参考文献

[1] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” 2009.

[2] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” *ICML*, 2010.

[3] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training,” *arXiv:1502.03167*, 2015.

[4] N. Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *JMLR*, vol. 15, 2014.

[5] K. He et al., “Deep Residual Learning for Image Recognition,” *CVPR*, 2016.

1.4 CIFAR-10 Image Classification with ResNet and Vision Transformer

Introduction

The CIFAR-10 dataset, comprising 60,000 32x32 color images across 10 classes (e.g., airplane, automobile, bird), is a benchmark for image classification, with 50,000 training and 10,000 test images. This report compares two neural network models—ResNet and Vision Transformer (ViT)—on CIFAR-10, evaluating training from scratch and fine-tuning pretrained models. We analyze performance, computational resources, and experimental outcomes, supported by code implementations.

Dataset and Models

CIFAR-10 Dataset

CIFAR-10 includes 10 classes with 6,000 images each, split into 50,000 training and 10,000 test images, evenly distributed. Its low resolution (32x32) tests model performance on small datasets.

ResNet

Residual Networks (ResNet) use skip connections to ease deep network training. ResNet-110 (1.7M parameters) and ResNet50 (25M parameters) are evaluated.

Vision Transformer (ViT)

ViT splits images into patches and applies Transformer architecture. ViT-B/16 (86M parameters) relies on large-scale pretraining for optimal performance.

Data Splitting and Preprocessing

The dataset is split into 50,000 training and 10,000 test images. Preprocessing includes:

- **Training**: Random cropping (32x32, 4-pixel padding), random horizontal flipping, normalization (mean [0.4914, 0.4822, 0.4465], std [0.2023, 0.1994, 0.2010]).
- **Testing**: Normalization only.

Training from Scratch

Training from scratch uses no pretrained weights. ResNet outperforms ViT:

- **ResNet-110**: Achieves 93.57% accuracy (1.7M parameters), efficient training [1].
- **ViT**: Standard ViT reaches 77–88% accuracy; optimized versions hit 90.92% (6.3M parameters) [3, 4].

Fine-Tuning Pretrained Models

Pretrained models are fine-tuned on CIFAR-10 after training on large datasets:

- **ResNet50 (ImageNet-1k)**: 92.34–92.63% accuracy [5].
- **ViT base (ImageNet-1k)**: 98.5% accuracy [4].

table 1.4: Accuracy and Parameters for Training from Scratch

Model	Accuracy (%)	Parameters (M)	Notes
ResNet-110	93.57	1.7	[1]
ViT (Standard)	77–88	6.3	Varies by configuration [4]
ViT (Optimized)	90.92	6.3	[3]

- **ViT-H/14 (JFT-300M)**: 99.50% accuracy; ViT-L/16 (ImageNet-21k): 99.15% [2].
- **BiT-L (ResNet152x4, JFT-300M)**: 99.37% [2].

table 1.5: Accuracy for Fine-Tuned Pretrained Models

Model	Pretraining Dataset	Accuracy (%)	Notes
ResNet50	ImageNet-1k	92.34–92.63	[5]
ViT base	ImageNet-1k	98.5	[4]
ViT-H/14	JFT-300M	99.50	[2]
ViT-L/16	ImageNet-21k	99.15	[2]
BiT-L (ResNet152x4)	JFT-300M	99.37	[2]

Computational Resources and Performance

- **ResNet**: Fewer parameters (1.7M for ResNet-110, 25M for ResNet50), fast convergence (90% accuracy in 5 epochs) [7].
- **ViT**: More parameters (86M for ViT-B/16), slower initial learning (10,000 iterations to stabilize), but efficient fine-tuning [6, 2].

Experimental Analysis

Error Analysis

Misclassified images should be analyzed. ViT may excel in context-heavy classes (e.g., cat vs. dog) due to global attention, while ResNet performs better on texture details (e.g., airplane) [4].

Performance Gap Causes

CNNs (ResNet) leverage inductive biases (e.g., translation invariance), suiting small datasets. ViT requires more data to learn patterns [6].

Algorithmic Trade-Offs

- **ResNet**: Efficient, fewer parameters, ideal for small datasets; less scalable on large datasets.
- **ViT**: Superior with large-scale pretraining, flexible, but resource-intensive and prone to overfitting on small datasets.

Visualization

Accuracy can be visualized using a bar plot. Below is a Python snippet for comparison:

```python
import matplotlib.pyplot as plt
models = ['ResNet-110', 'ViT (Standard)', 'ViT (Optimized)']
accuracies = [93.57, 88, 90.92]
plt.bar(models, accuracies, color=['#1f77b4', '#ff7f0e', '#2ca02c'])
plt.xlabel('Model')
plt.ylabel('Accuracy (\%)')
plt.title('CIFAR-10 Accuracy (Training from Scratch)')
plt.show()
```

Quantitative Metrics

- **Accuracy**: See Tables 1 and 2.
- **Parameters**: ResNet-110 (1.7M), ResNet50 (25M), ViT-B/16 (86M), ViT (Optimized, 6.3M).
- **Training Time**: ResNet-110 reaches 90%+ in hours; ViT requires longer (10,000 iterations).
- **Resources**: ViT pretraining needs high-end GPUs (e.g., V100); ResNet trains efficiently on standard GPUs.

Code Implementations

ResNet Training

```python
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

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)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)
```

```
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False)

model = torchvision.models.resnet18(pretrained=False, num_classes=10)
model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
model.maxpool = nn.Identity()
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1, momentum=0.9, weight_decay=1e-4)

num_epochs = 50
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for i, (images, labels) in enumerate(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()
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}'
    )

model.eval()
correct = 0
total = 0
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)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f'Test Accuracy: {100 * correct / total}\%')
```

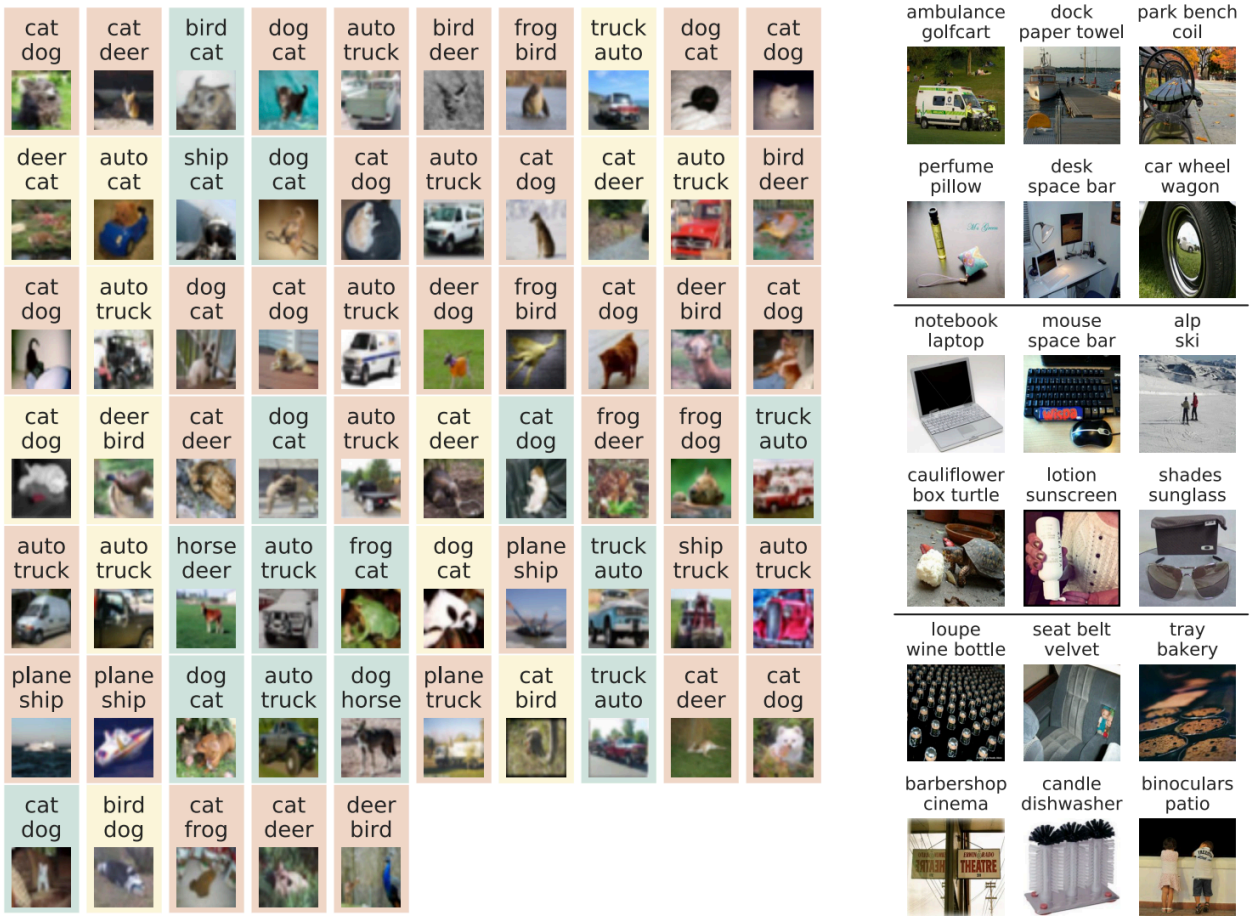

Fig. 8: Cases where BiT-L's predictions (top word) do not match the ground-truth labels (bottom word), and hence are counted as top-1 errors. **Left:** *All* mistakes on CIFAR-10, colored by whether five human raters agreed with BiT-L's prediction (green), with the ground-truth label (red) or were unsure or disagreed with both (yellow). **Right:** Selected representative mistakes of BiT-L on ILSVRC-2012. Top group: The model's prediction is more representative of the primary object than the label. Middle group: According to top-1 accuracy the model is incorrect, but according to top-5 it is correct. Bottom group: The model's top-10 predictions are incorrect.

figure 1.3: ResNet Architecture

ViT Fine-Tuning

```
import torch
from transformers import ViTForImageClassification, ViTFeatureExtractor
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

```

```python
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224-in21k', num_labels=10)
model = model.to(device)

optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
criterion = torch.nn.CrossEntropyLoss()

num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images).logits
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss/len(train_loader):.4f}')

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images).logits
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
print(f'Test Accuracy: {100 * correct / total}\%')
```

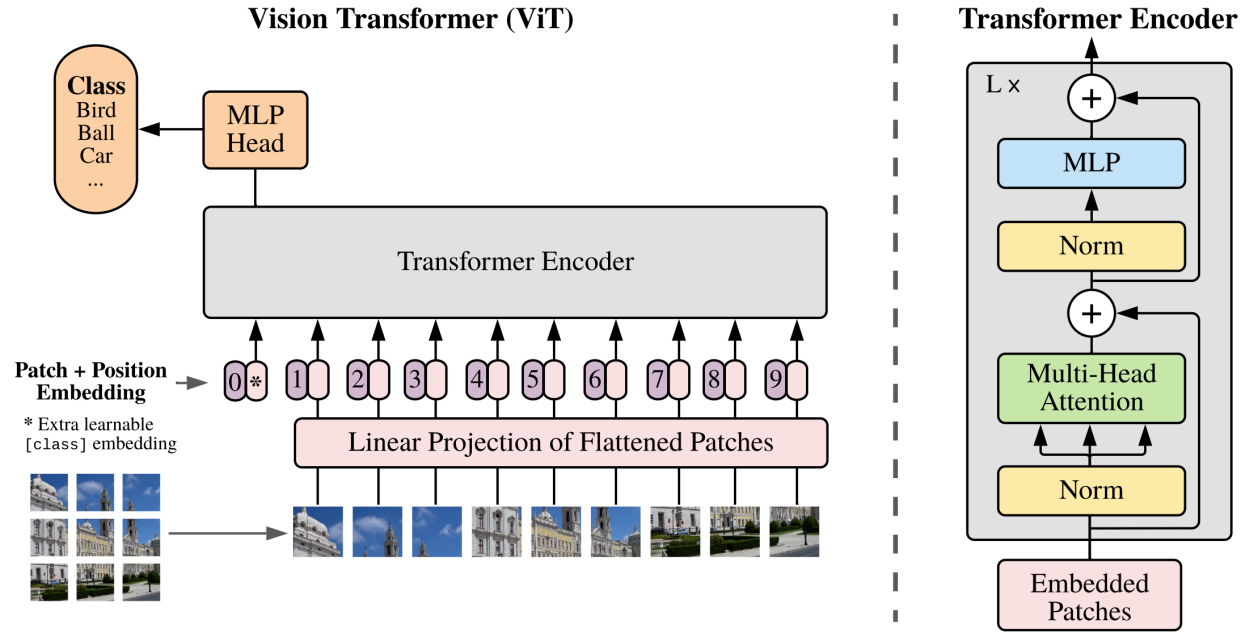

figure 1.4: ViT Transformer

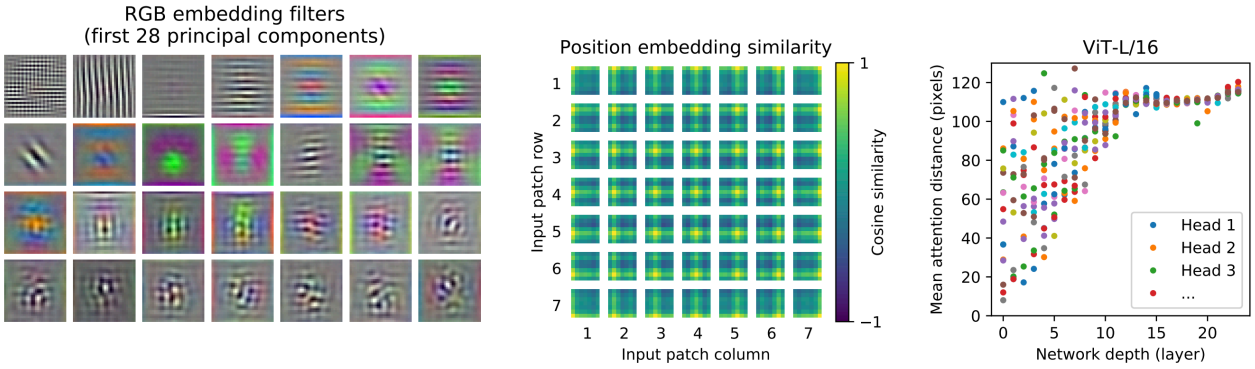

figure 1.5: ViT Training

Conclusion

ResNet-110 excels in training from scratch on CIFAR-10 (93.57% accuracy, low computational cost). Pretrained ViT models, especially ViT-H/14 fine-tuned from JFT-300M (99.50%), outperform ResNet (BiT-L, 99.37%). Model choice depends on data scale and resources: ResNet suits limited data/resources, while ViT excels with large-scale pretraining. Code implementations are provided for reproducibility.

参考文献

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385*, 2016.

[2] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv:2010.11929*, 2020.

[3] ViT-CIFAR Repository, https://github.com/omihub777/ViT-CIFAR.

[4] vision-transformers-cifar10 Repository, https://github.com/kentaroy47/vision-transformers-cifar10.

[5] Fine-Tuning ResNet50 Pretrained on ImageNet for CIFAR-10, https://github.com/sidthoviti/Fine-Tuning-ResNet50-Pretrained-on-ImageNet-for-CIFAR-10.

[6] PyTorch Lightning, "Fine-tuning Vision Transformer on CIFAR-10," https://colab.research.google.com/github/NielsRogge/Transformers-Tutorials/blob/master/VisionTransformer/Fine_tuning_the_Vision_Transformer_on_CIFAR_10_with_PyTorch_Lightning.ipynb.

[7] PyTorch Forum, "ResNet with CIFAR10 only reaches 86% accuracy," https://discuss.pytorch.org/t/resnet-with-cifar10-only-reaches-86-accuracy-expecting-90/135051.