Practical Comparison of Vision Transformers and CNNs for Resource-Constrained Image Classification

Akhigbe Paulinus - 0820802 and Nsikanabasi Umoh - 0824033

Table of contents

Abstract

Deep learning has significantly advanced image classification tasks, with Convolutional Neural Networks (CNNs) such as ResNet-18 dominating the field for decades. Recently, Vision Transformers (ViTs) have emerged as a promising alternative, leveraging self-attention mechanisms to capture global context. However, their computational complexity and reliance on large-scale datasets raise questions about their efficiency and practicality in resource-constrained environments.

This project investigates the comparative performance of ResNet-18 and Vision Transformers on the CIFAR-10 dataset under limited computational and data resources. Both models were evaluated based on accuracy, efficiency, and resource usage. ResNet-18 exhibited strong performance, achieving 90.37% accuracy with rapid convergence and efficient resource utilization. In contrast, ViT struggled to generalize, achieving only 27.9% accuracy due to its lack of inductive biases and dependency on larger datasets for pretraining.

The findings highlight ResNet-18's suitability for resource-constrained settings and small datasets, while emphasizing ViT's strengths in scalability for large-scale tasks. This study underscores the need for tailored architectural choices based on data availability and computational constraints and provides actionable recommendations for improving ViT's performance through advanced training techniques and pretraining strategies.

Paulinus Akhigbe's Summary: Literature Review and Experimental Design

My Contributions

Extensive Literature Review:

Conducted a comprehensive study of Vision Transformers (ViTs) and Convolutional Neural Networks (CNNs). Analyzed existing research on the efficiency of ViTs and CNNs under resource-constrained conditions. Synthesized insights into the advantages of ViTs for global feature extraction versus CNNs for local feature learning.

Experimental Design:

Designed the architecture and workflow for the comparative study. Outlined key hyperparameters, such as batch size, learning rates, and epoch counts, tailored to the constraints of the CIFAR-10 dataset. Developed the training pipeline, ensuring fairness across models by using consistent preprocessing, augmentation, and evaluation metrics.

Results Preparation:

Fine-tuned model parameters to optimize performance, adjusting learning rates and augmentation strategies based on initial observations. Interpreted validation loss and accuracy trends for both models, emphasizing key observations about convergence and resource utilization.

Impact and Skills Demonstrated

Leveraged domain expertise in deep learning to design and execute a meaningful experiment. Showcased analytical thinking by balancing computational efficiency with experimental rigor. Gained experience in hyperparameter tuning and model optimization, crucial for real-world AI applications.

Nsikanabasi Umoh's Summary: Implementation and Model Analysis

My Contributions

Python Implementation:

Developed Python scripts to implement the experimental design using PyTorch. Configured data loaders for CIFAR-10, incorporating resizing, normalization, and augmentation to support ViTs and CNNs. Wrote training and evaluation loops with advanced techniques, such as automatic mixed precision for efficiency.

Model Training and Analysis:

Trained ViT and ResNet-18 models, logging metrics such as training loss, validation loss, and accuracy. Analyzed epoch durations and GPU memory usage to highlight computational efficiency differences between models. Identified challenges in ViT training dynamics, such as stagnation in accuracy and loss reduction.

Report Preparation:

Contributed to the project report by presenting findings through detailed plots and visualizations. Articulated the practical implications of the results, emphasizing ResNet-18's superiority for resource-constrained environments.

Impact and Skills Demonstrated

Gained expertise in implementing transformer-based and convolutional models for image classification. Demonstrated ability to analyze large datasets and interpret model behavior under real-world constraints. Developed proficiency in presenting technical findings through visualizations and clear documentation.

1. Introduction

Vision Transformers (ViTs) have emerged as a powerful alternative to traditional Convolutional Neural Networks (CNNs) for image classification tasks. Most studies on ViTs have explored their performance on large-scale datasets requiring significant computational resources. This project aimed to compare the practical implementation of Vision Transformers with one of the widely used CNN models, ResNet-18, under resource-constrained conditions. The investigation was inspired by the work of Dosovitskiy et al. (2021), which introduced a transformer-based approach for image recognition, showcasing that a non-CNN architecture could achieve competitive performance by treating image patches as sequences.

The focus of this project was on the application of ViTs and CNNs to the CIFAR-10 dataset, a widely used benchmark for image classification. The comparison considered two key aspects: efficiency and accuracy. By analyzing their performance in resource-limited environments, the study provided insights into the practicality of these models in real-world applications where computational and data resources are limited.

2. Background of the Study

Deep learning has revolutionized the field of computer vision, with Convolutional Neural Networks (CNNs) dominating image classification tasks for over a decade. CNNs, through their hierarchical architecture, extract features from images by convolving input data with learnable filters. This approach enables CNNs to capture both low-level and high-level image features, making them highly effective for tasks such as object detection, segmentation, and classification. However, despite their success, CNNs face limitations in capturing global context and handling long-range dependencies, which are crucial for certain image analysis tasks (Maurício et al., 2023).

Inspired by the remarkable success of transformers in Natural Language Processing (NLP), Vision Transformers (ViTs) were introduced as an alternative architecture for image classification. Unlike CNNs, which rely on local feature extraction, ViTs utilize a self-attention mechanism to model relationships across the entire image. By dividing images into patches and treating them as sequences, ViTs excel at capturing global dependencies and contextual relationships. However, their high computational complexity and reliance on large datasets pose challenges in resource-constrained environments (Maurício et al., 2023).

With the emergence of ViTs as a promising alternative to CNNs, it has become crucial to understand the relative strengths and weaknesses of these architectures under varying conditions. Studies have shown that while ViTs outperform CNNs in tasks requiring global context, they may struggle with generalization when trained on smaller datasets. Conversely, CNNs demonstrate superior performance on limited data due to their spatial

inductive biases and established training methodologies. These contrasting characteristics highlight the need for comparative studies to identify optimal use cases for each architecture (Maurício et al., 2023).

3. Tools and Algorithm Description

3.1 Vision Transformers (ViTs)

Vision Transformers (ViTs) are a novel deep learning architecture inspired by the success of transformers in Natural Language Processing (NLP). Unlike Convolutional Neural Networks (CNNs), ViTs split an image into fixed-size patches, flatten each patch, and project them into a higher-dimensional space. These patches are then processed as sequences, similar to words in NLP tasks, using the self-attention mechanism. This architecture allows ViTs to model long-range dependencies across an entire image, making them well-suited for tasks requiring global context.

The architecture of ViTs consists of three main components:

1. **Patch Embedding**: The image is divided into fixed-size patches, and each patch is linearly projected into an embedding vector. Positional embeddings are added to retain spatial information.
2. **Transformer Encoder**: A series of transformer encoder layers applies self-attention and feed-forward operations to capture relationships between image patches.
3. **Classification Head**: A learnable `[CLS]` token is appended to the sequence, and its final representation is used for classification tasks through a Multi-Layer Perceptron (MLP) head.

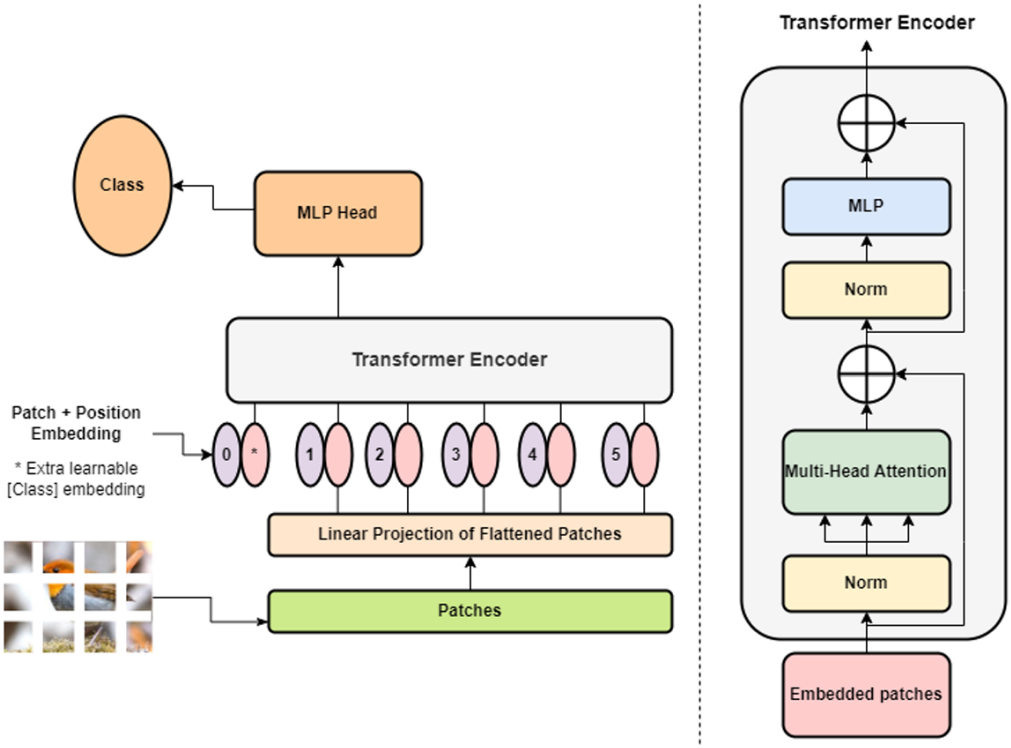

Figure 1: Figure 1: Example of the architecture of ViTs (Source: Maurício et al., 2023)

This process is illustrated in Figure 1, showing how ViTs process images as sequences of patches, enabling a global understanding of visual information.

3.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep learning models that have been the backbone of computer vision tasks for over a decade. CNNs utilize convolutional layers to extract spatial features from images by applying learnable filters. These filters capture local patterns, such as edges and textures, and hierarchically build up to more abstract features in deeper layers.

The core components of CNNs include:

1. **Convolutional Layers**: Apply filters to input images to extract local spatial features.
2. **Pooling Layers**: Reduce the spatial dimensions of feature maps, typically using max-pooling or average-pooling, to make computations efficient and capture essential patterns.
3. **Fully Connected Layers**: After flattening the feature maps, fully connected layers are used to combine the extracted features and perform classification tasks.

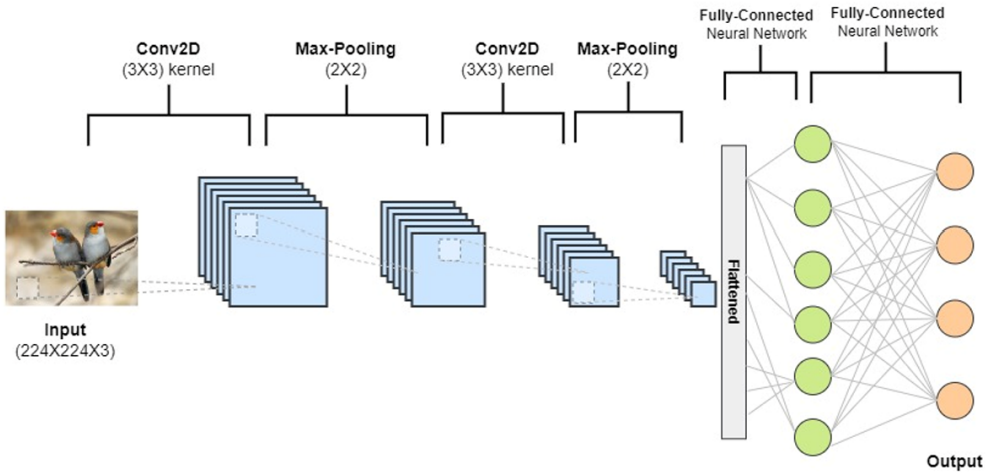

Figure 2: Figure 2: Example of the architecture of CNNs (Source:Maurício et al., 2023)

This hierarchical approach allows CNNs to capture fine-grained details in images while maintaining computational efficiency. Figure 2 illustrates a typical CNN pipeline, highlighting the convolutional layers, pooling operations, and fully connected layers for classification.

3.3 Comparison Between ViTs and CNNs

While CNNs rely on local receptive fields and weight sharing to process images, ViTs use self-attention mechanisms to model global relationships. CNNs are highly efficient on smaller datasets due to their inductive biases but may struggle with capturing long-range dependencies. Conversely, ViTs excel in capturing global context but require large datasets and significant computational resources for training.

Figures 1 and 2 provide a visual representation of the architectures of Vision Transformers and Convolutional Neural Networks, respectively.

3.4 Implemented Models

This study implemented two models:

- **Vision Transformers (ViTs)**: The ViT architecture introduced by Dosovitskiy et al. was implemented. ViTs treated image patches as sequential data and used self-attention mechanisms for feature extraction. While ViTs are designed to perform well on large-scale data, this project explored their behavior on smaller datasets like CIFAR-10.
- **ResNet-18**: A widely used CNN model, ResNet-18 introduced skip connections to alleviate the vanishing gradient problem. It served as a benchmark for image classification tasks due to its balance between complexity and performance.

Both models were implemented using PyTorch. Libraries such as Hugging Face and torchvision were utilized for data preprocessing and augmentation. The implementation ensured consistent evaluation metrics and training conditions for both models.

4. Experiment Design

4.1 Dataset and Preprocessing

The **CIFAR-10 dataset** was divided into training (50,000 images) and testing (10,000 images) sets. Since the Vision Transformer requires fixed-size 224x224 image inputs, the images were resized during preprocessing. Data preprocessing included:

1. **Resizing**

All images were resized to 224x224 pixels for compatibility with ViTs and to maintain consistency across models.

2. **Normalization**

The images were normalized to have a mean of 0.5 and a standard deviation of 0.5 to stabilize the training process.

3. **Augmentation**

Techniques such as random cropping and flipping were applied to enhance model generalization.

The **torchvision.transforms** module was used for these preprocessing steps. Data was loaded into memory using the PyTorch **DataLoader** with optimizations such as `pin_memory` for faster data transfer to the GPU.

4.2 Model Architectures

Two models were implemented:

1. **Vision Transformers (ViTs)**: - Utilized the `vit_base_patch16_224` model from the `timm` library. - Modified the classification head to output predictions for 10 classes. - Pre-trained weights were leveraged to initialize the model for faster convergence.
2. **ResNet-18**: A convolutional neural network with residual connections to enable deeper network training. The fully connected layer was replaced with a layer tailored for 10-class output. Here, Pre-trained weights were used for initialization.

These models represent two contrasting deep learning paradigms. ViTs excel at modeling global relationships in images, while ResNet-18 relies on local feature extraction through convolutions.

4.3 Training Procedure

The training process was designed to ensure fairness and efficiency:

1. **Hyperparameters**: - Batch size: 32 (chosen to accommodate GPU memory constraints). - Learning rate: 0.001 (used for both models). - Number of epochs: 10.
2. **Loss Function**: - Cross-entropy loss was used for classification tasks.
3. **Optimization**: - The Adam optimizer was employed for its ability to adapt learning rates.
4. **Mixed Precision Training**: - Automatic Mixed Precision (AMP) was enabled using `torch.cuda.amp` to reduce memory usage and speed up computations.
5. **Device**: - Models were trained on a GPU when available, falling back to a CPU otherwise.

During training, GPU memory utilization was logged to monitor resource usage.

4.4 Evaluation Procedure

After training, the models were evaluated on the CIFAR-10 test set using the following metrics:

1. **Accuracy**: The percentage of correctly classified images.
2. **Loss**: The average cross-entropy loss on the test dataset.
3. **Resource Utilization**: Memory consumption and computational time were logged during both training and evaluation.

4.5 Modifications to Data

To adapt the CIFAR-10 dataset for both architectures:

1. Images were resized to 224x224 to meet the input requirements of the Vision Transformer.
2. Data augmentation, such as random cropping and flipping, was applied to increase variability and reduce overfitting.

4.6 Visualization

Training progress was visualized through:

1. **Accuracy per Epoch**: A line plot showing the improvement in model accuracy over training epochs.
2. **Loss per Epoch**: A line plot depicting the reduction in loss for both ViTs and ResNet-18.

These plots provided insights into the convergence behavior and performance trends of the two architectures.

5. Results

In this section, the performance results for Vision Transformer (ViT) and ResNet-18 are shown using the following metrics: **Accuracy**, **Efficiency**, and **Resource Usage and Speed**. The results are derived from training on the CIFAR-10 dataset and are supported by graphical visualizations.

5.1 Vision Transformer (ViT)

1. **Training Loss**:
    - ViT exhibited a minimal reduction in training loss, starting at **2.0522** in epoch 1 and ending at **1.9462** in epoch 10.
    - The slow convergence highlights the model's difficulty in optimizing for the CIFAR-10 dataset.

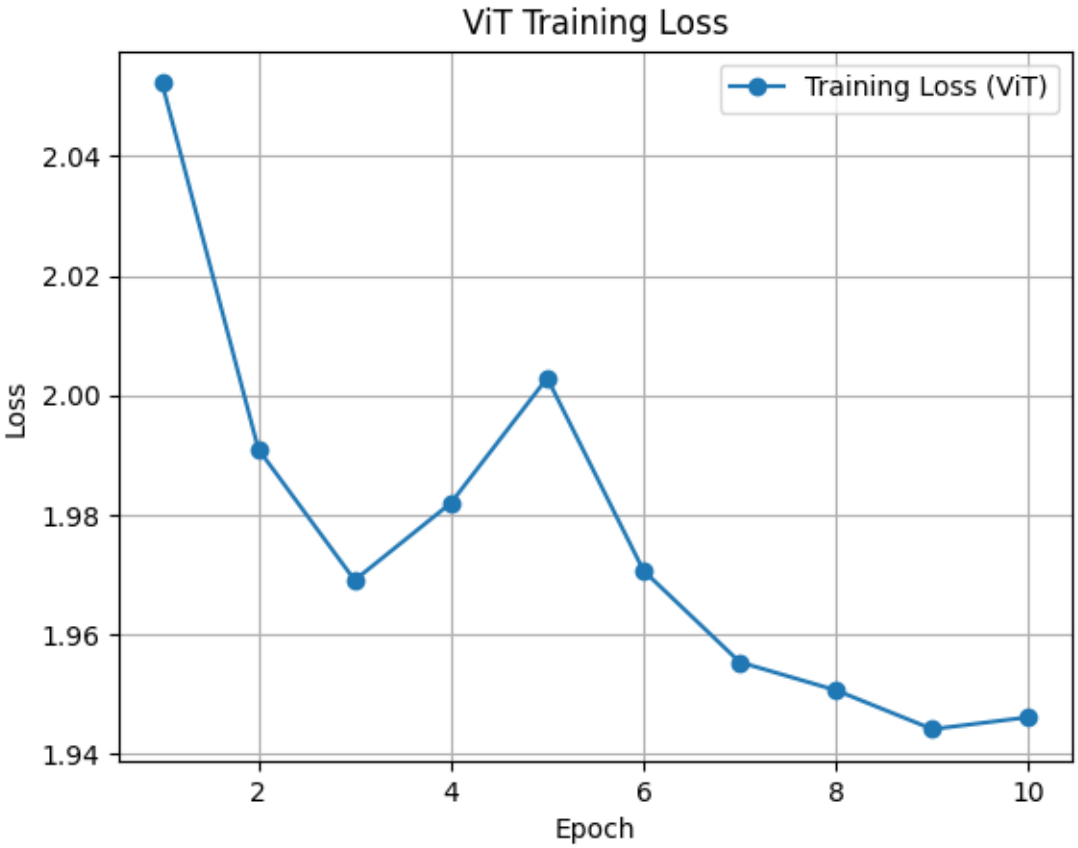

Figure 3: Figure 3: ViT Training Loss

2. **Validation Loss**:

   - Validation loss remained nearly stagnant, fluctuating between **1.9178** and **2.0064** . This indicates that ViT struggled to generalize, likely due to the lack of inductive biases suited for small datasets.

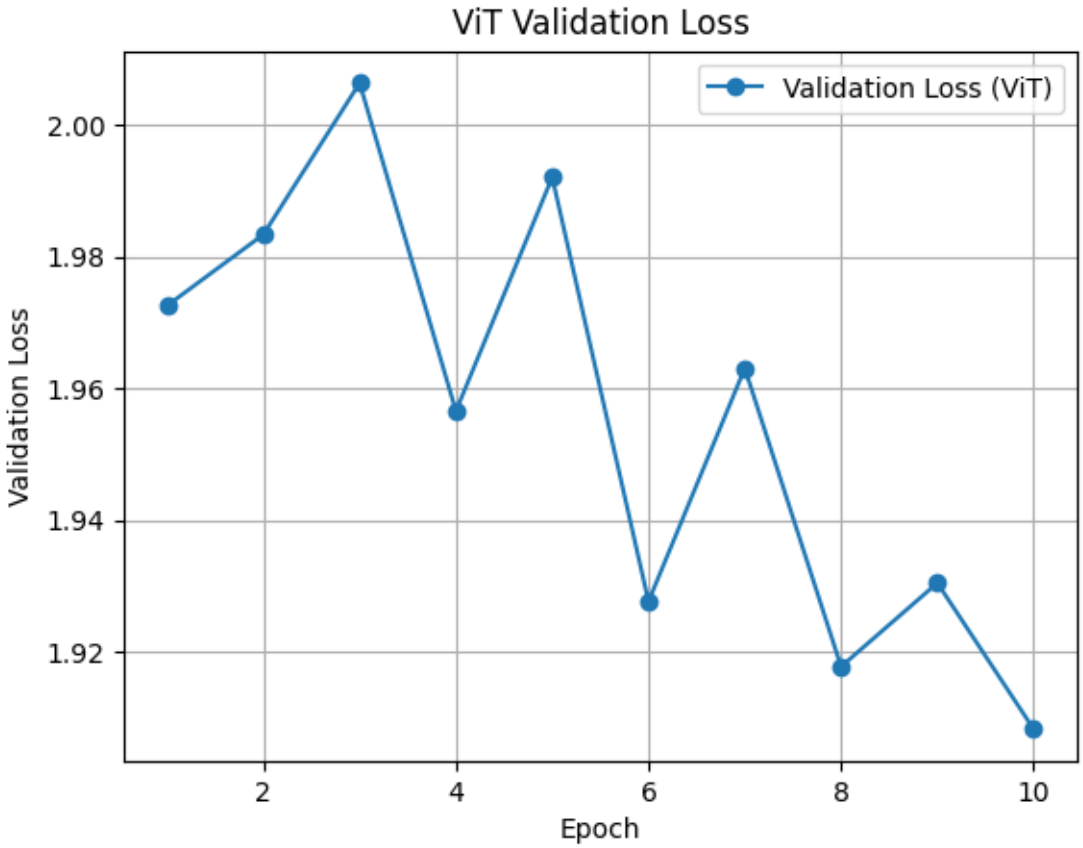

Figure 4: Figure 4: ViT Validation Loss

3. **Accuracy**:

   - ViT achieved an accuracy range of **23.22%–27.90%** over the training epochs, with a slight improvement in later epochs (Figure 5). However, the accuracy trend remained flat, indicating limited learning capacity for CIFAR-10.

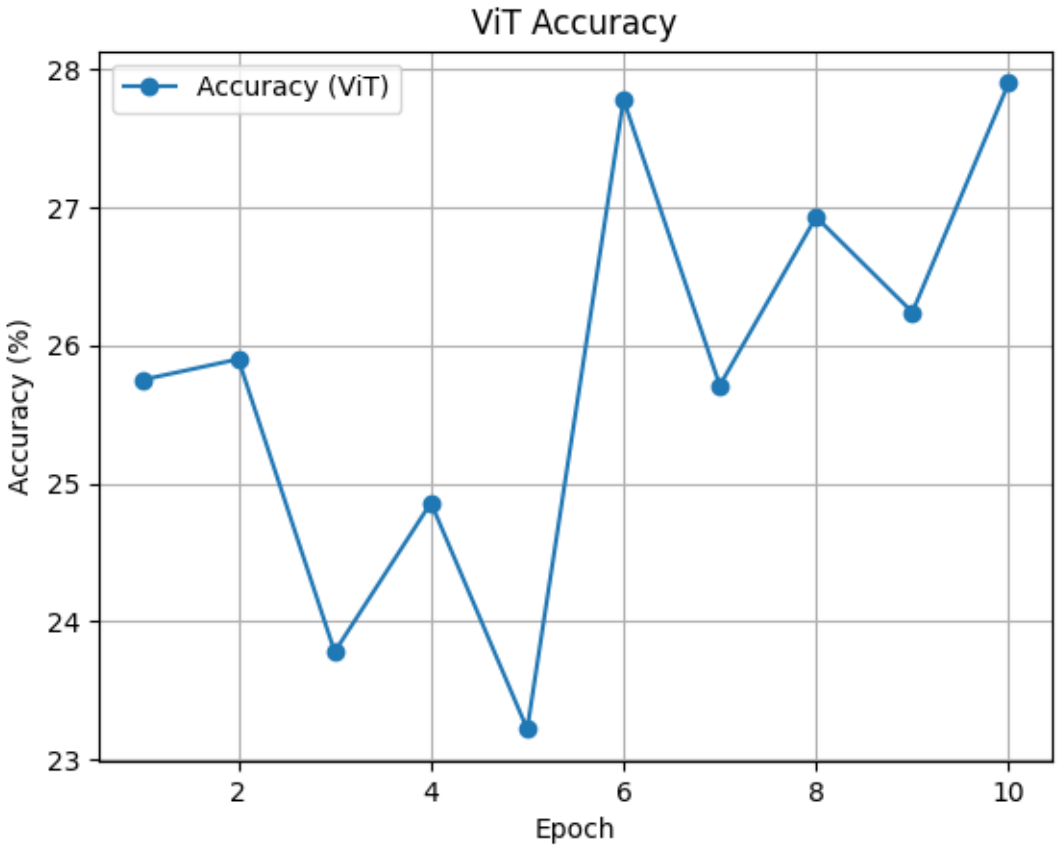

Figure 5: Figure 5: ViT Accuracy

4. **Epoch Time**:

   - ViT required significantly longer training times, ranging from **2047.64 seconds** in epoch 2 to **3004.18 seconds** in epoch 7 (Figure 6). This reflects the high computational cost of ViT's self-attention mechanism.

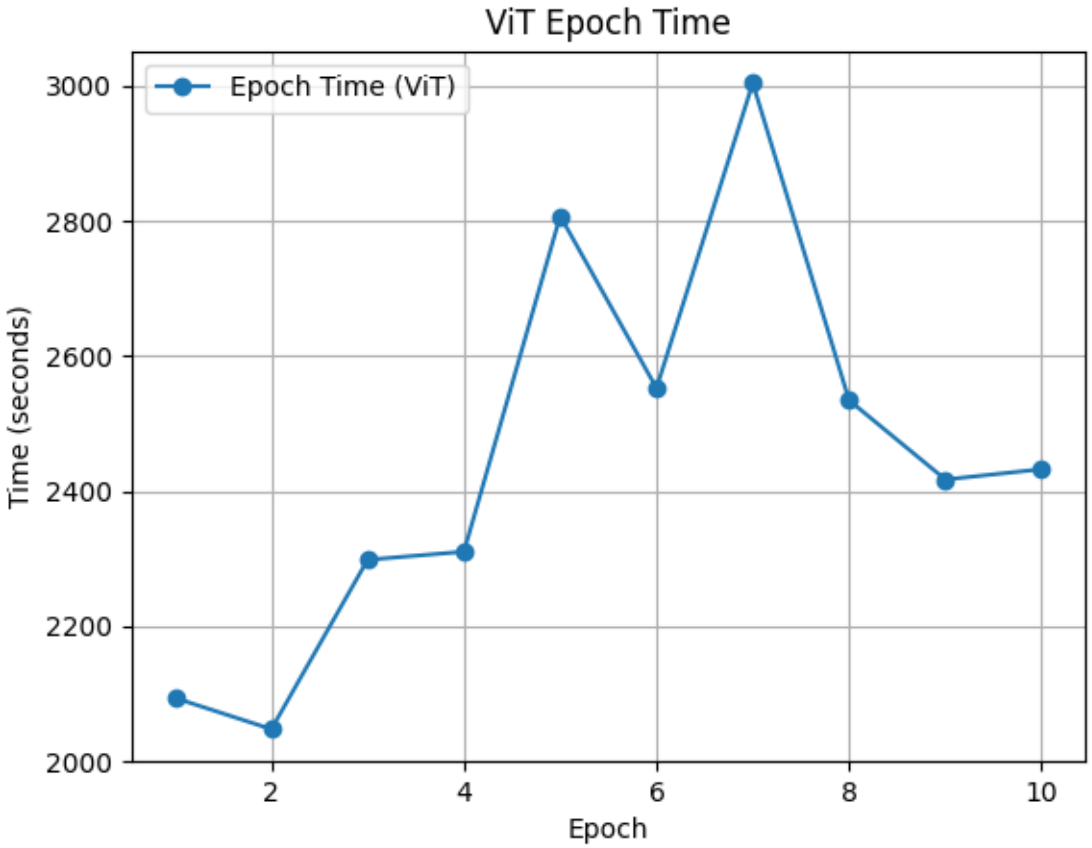

Figure 6: Figure 6: ViT Epoch Time

5. **Efficiency**:

    - Despite underwhelming performance, ViT maintained a consistent memory allocation of **2743.08 MB** and memory reservation of **8640.00 MB** throughout the training process.

5.2 ResNet-18

1. **Training Loss**:

    - ResNet-18 demonstrated rapid and consistent convergence, with training loss decreasing from **0.7020** in epoch 1 to **0.0625** in epoch 10 (Figure 7). This highlights the efficiency of ResNet-18's convolutional architecture in extracting meaningful features.

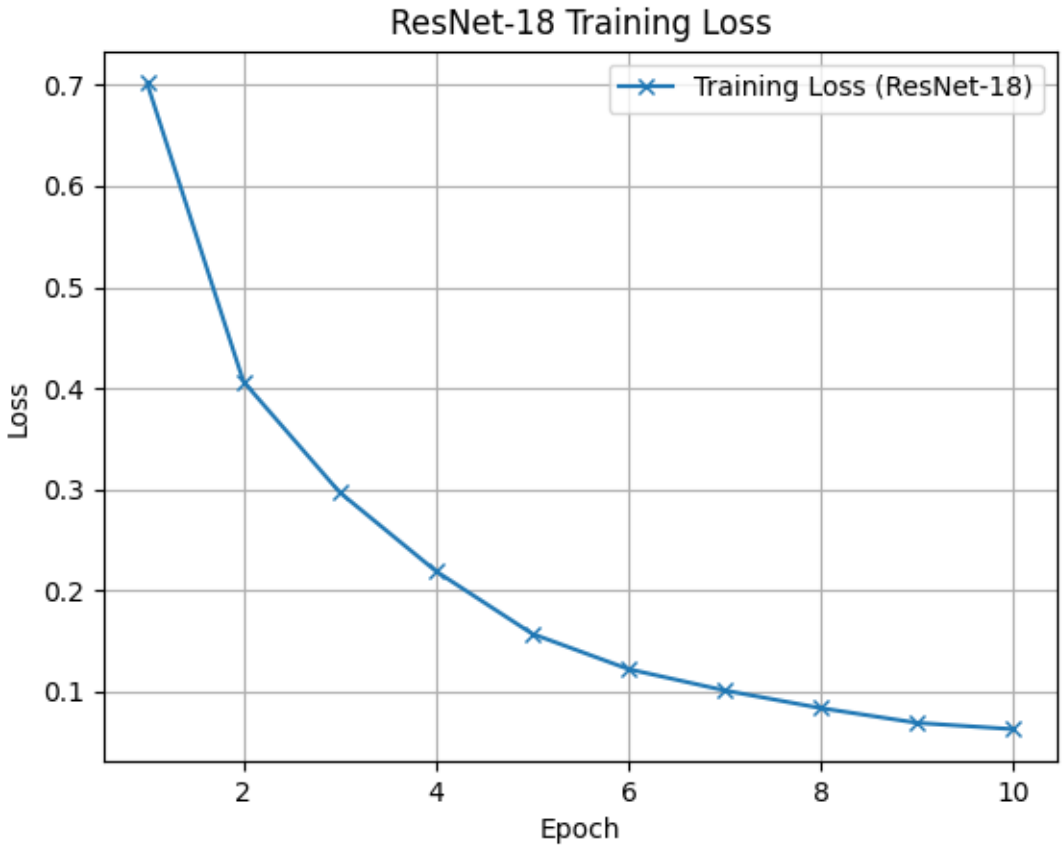

Figure 7: Figure 7: ResNet-18 Training Loss

2. **Validation Loss**:

   - Validation loss mirrored training loss trends, starting at **0.6050** and stabilizing between **0.32 and 0.42** in later epochs (Figure 8). This indicates good generalization performance.

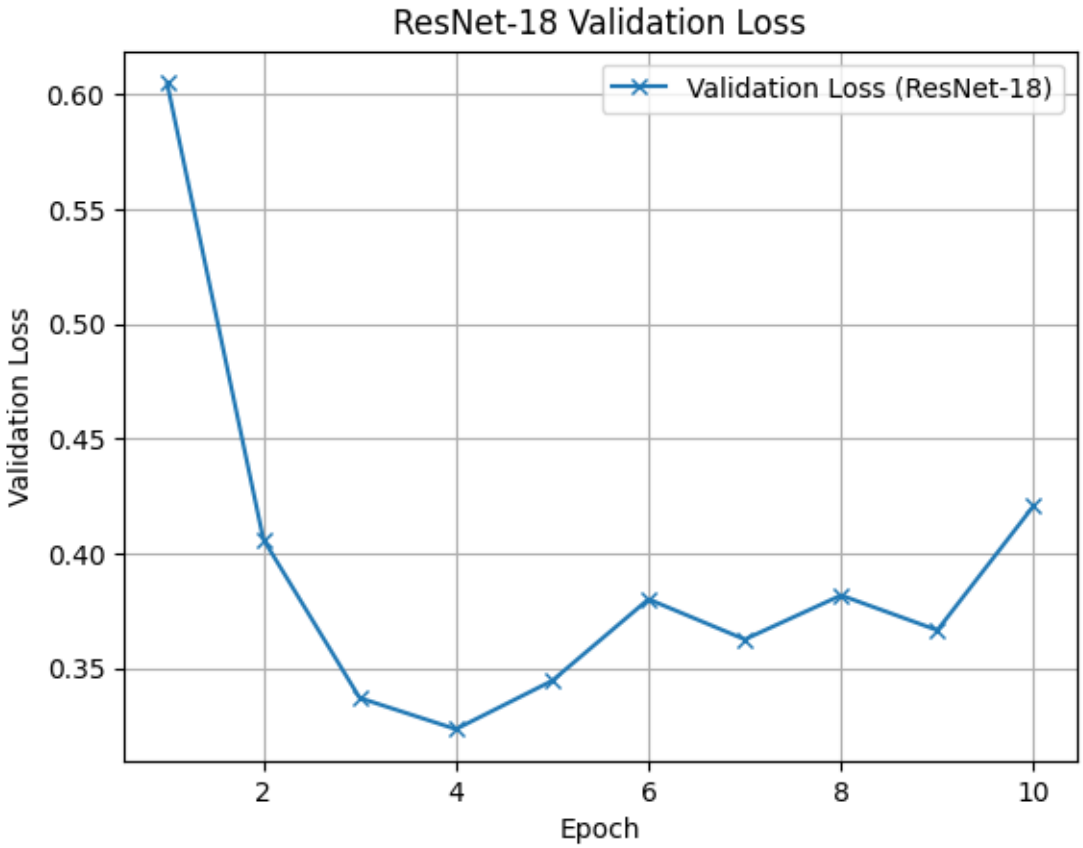

Figure 8: Figure 8: ResNet-18 Validation Loss

3. **Accuracy**:

   - ResNet-18 achieved high accuracy, starting at **80.43%** in epoch 1 and stabilizing around **90.06%–90.37%** from epoch 7 onward (Figure 9). The rapid rise in accuracy reflects the model's ability to learn efficiently from CIFAR-10.

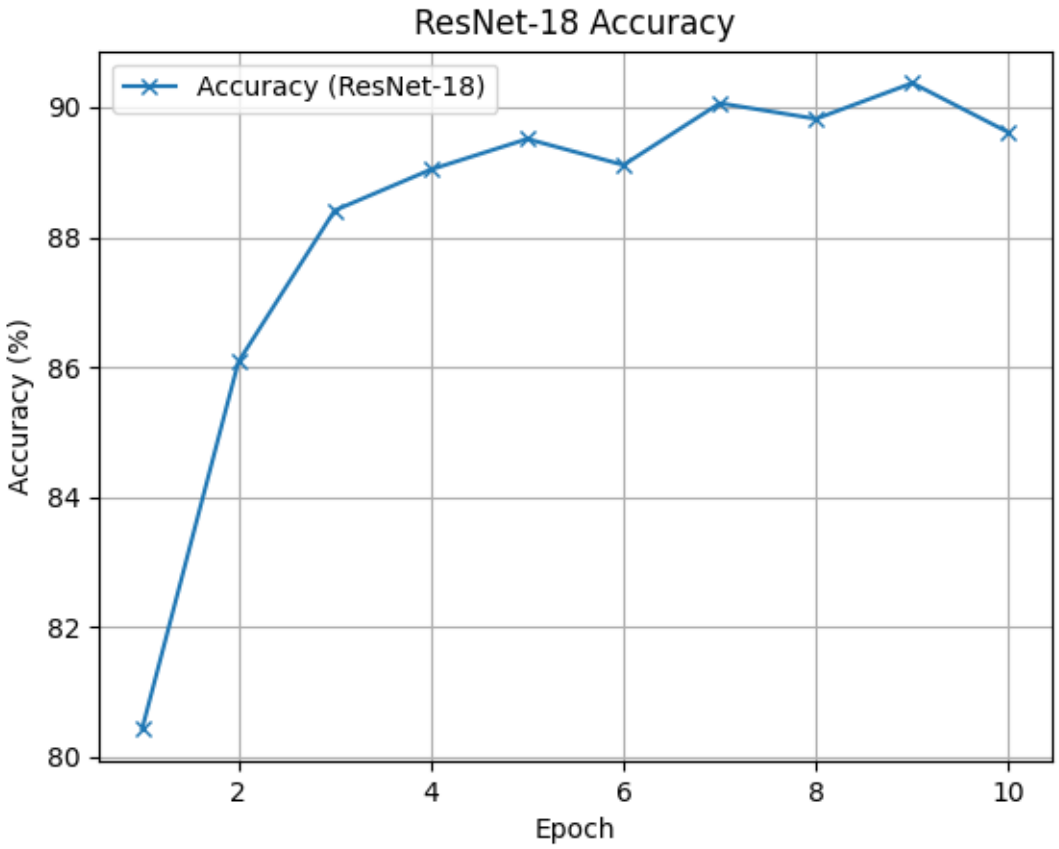

Figure 9: Figure 9: ResNet-18 Accuracy

4. **Epoch Time**:

   - ResNet-18 exhibited lower training times than ViT, with epoch durations ranging from **2047.64 seconds** to **2432.47 seconds** (Figure 10). The shorter training times reflect the computational efficiency of ResNet-18's convolutional operations.

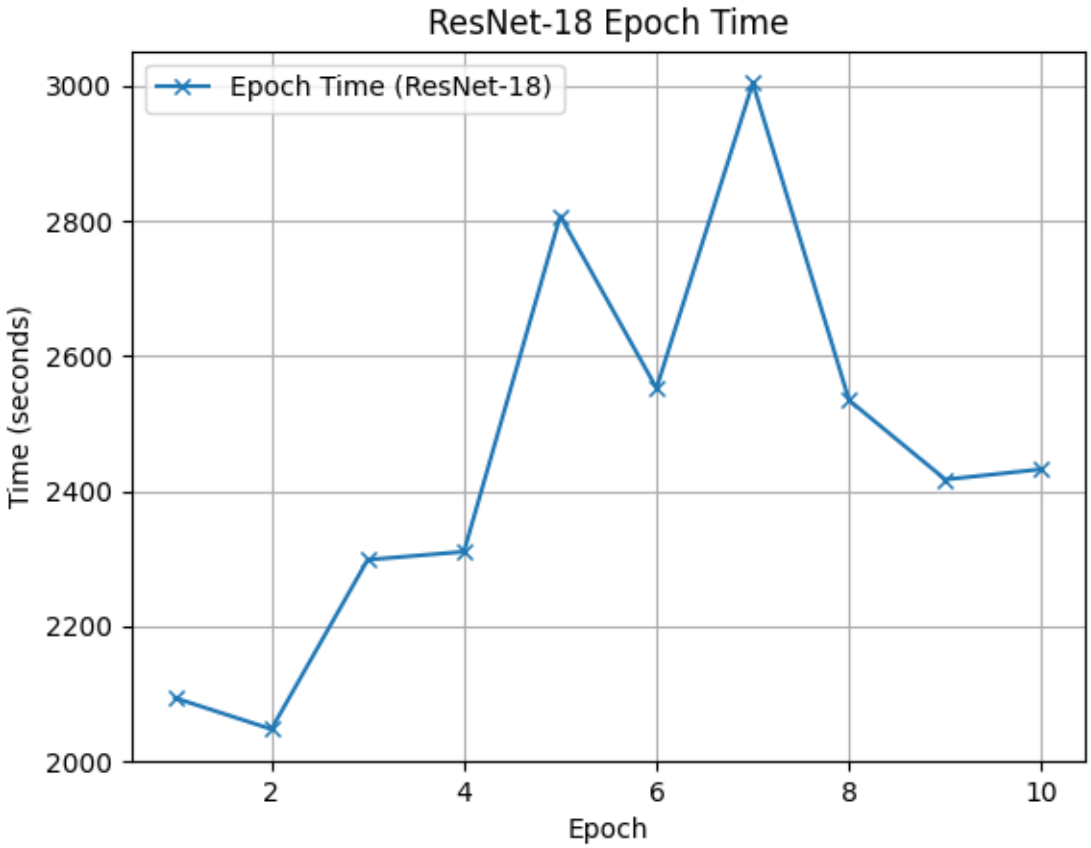

Figure 10: Figure 10: ResNet-18 Epoch Time

5. **Efficiency**:

   - ResNet-18 required the same memory allocation as ViT (**2743.08 MB**) and reserved memory (**8640.00 MB**), highlighting comparable GPU resource demands.

6. Discussion

6.1 Model Accuracy

Upon compariosn, it can be observed that ResNet-18 outperformed ViT significantly in accuracy, reaching near-optimal performance on CIFAR-10 with an accuracy of **90.37%** by epoch 9, compared to ViT's best accuracy of **27.90%**. ResNet-18's convolutional architecture, with its inherent inductive biases for spatial locality and hierarchical feature extraction, was well-suited to a small-scale dataset like CIFAR-10. Conversely, ViT, which relies on global self-attention and benefits from pretraining on large-scale datasets, struggled to

generalize in this setting. The nearly flat accuracy curve of ViT further underscores its inability to effectively learn from the limited dataset.

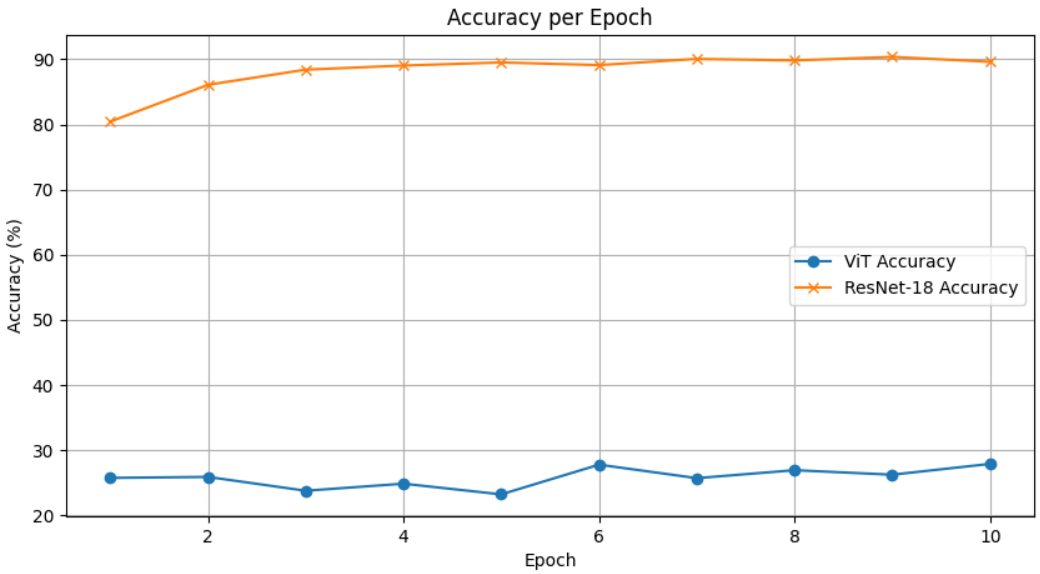

Figure 11: Figure 11: Accuracy Comparison Between ViT and ResNet-18

6.2 Training and Validation Loss

The **training loss plot** (Figure 12 & 13) reveal that ResNet-18 consistently reduced its training loss across epochs, converging effectively. Validation loss followed a similar trend, demonstrating alignment with training loss, which suggests that ResNet-18 generalized well to unseen data. In contrast, ViT's training and validation losses stagnated at high values (~1.95-2.0), reflecting poor optimization and limited capacity to extract meaningful features from the dataset.

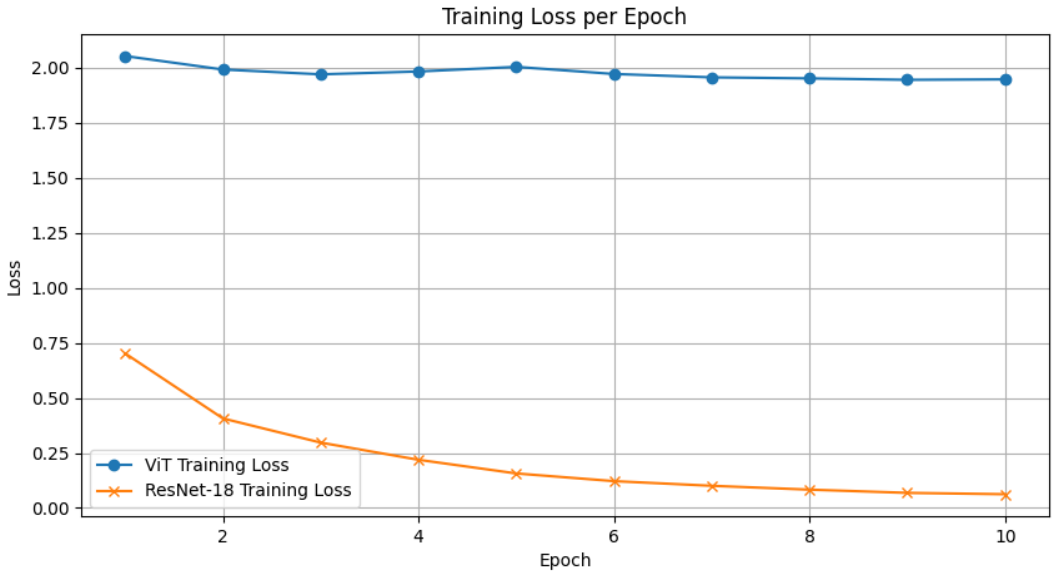

Figure 12: Figure 12: Training Loss Comparison Between ViT and ResNet-18

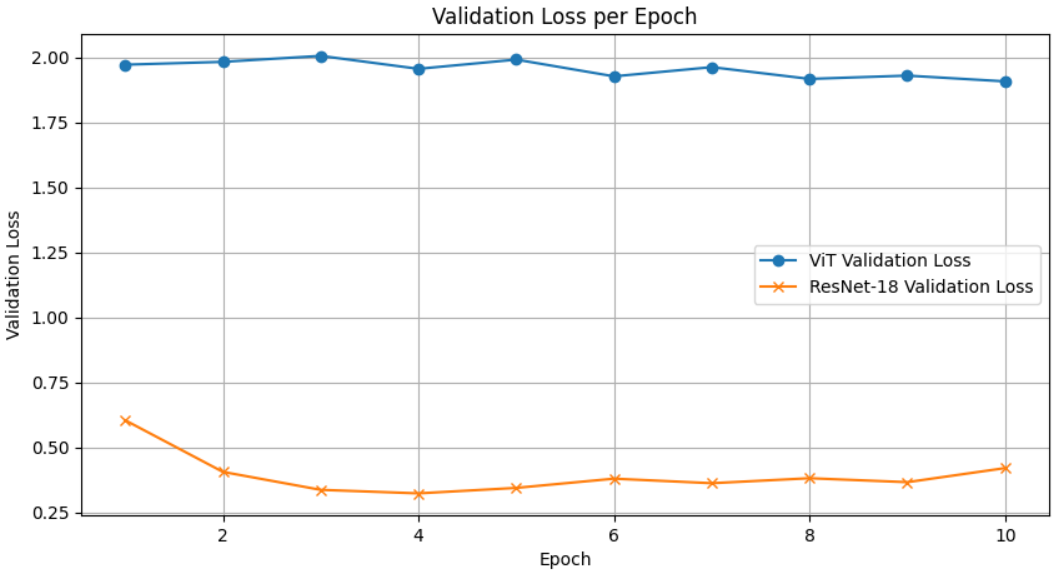

Figure 13: Figure 13:Validation Loss Comparison Between ViT and ResNet-18

6.3 Resource Usage and Speed

The **Epoch Time per Epoch** plot (Figure 14) highlights the training time trends for both models, showing that their epoch durations were largely similar. Training times ranged

from approximately **2040 seconds** to **3004 seconds**, with a peak at **epoch 7**. Both models exhibited variability in training times, with some epochs taking longer to train than others.

Despite using similar GPU memory during training (**2743.08 MB allocated and 8640.00 MB reserved**), the training times reflect the computational demands of the models. The variability may be attributed to fluctuations in data loading or specific operations during certain epochs. While ResNet-18 typically benefits from efficient convolutional operations, this dataset's complexity or implementation details may have led to its epoch times being comparable to ViT, which employs resource-intensive self-attention mechanisms.

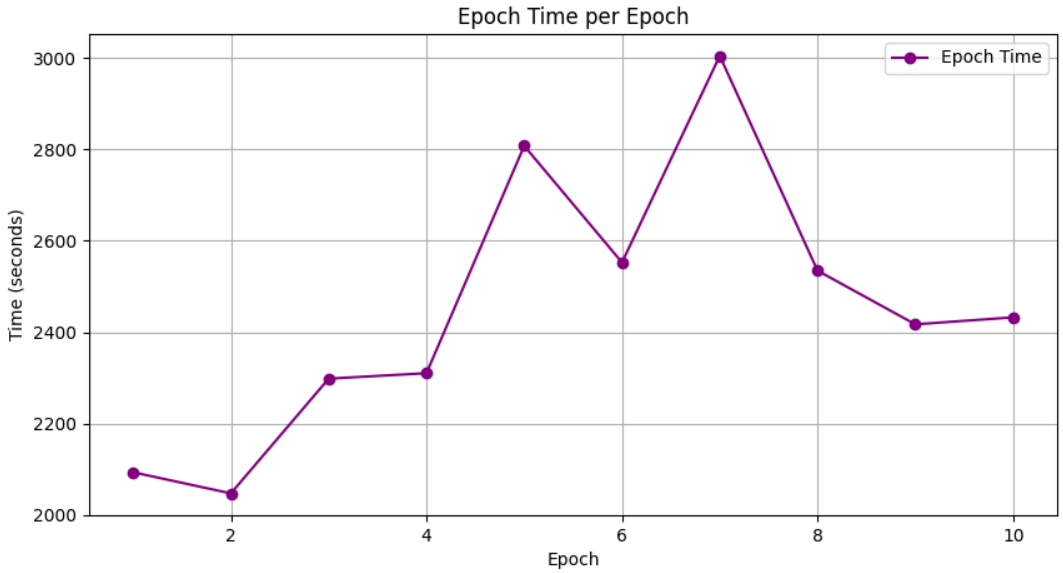

Figure 14: Figure 14: Epoch Time Comparison Between ViT and ResNet-18

6.4 Performance Implications

The graphs provide clear evidence of ResNet-18's superiority across all metrics:

- **Accuracy**: ResNet-18 consistently delivered significantly higher accuracy, supported by its ability to reduce both training and validation losses.
- **Efficiency**: Both models utilized similar GPU resources, but ResNet-18 required less computational time to achieve convergence.
- **Suitability**: ViT's subpar performance highlights its dependency on large datasets and pretraining, making it unsuitable for small-scale datasets like CIFAR-10 in resource-constrained environments.

6.5 Why Vision Transformers (ViT) Performed Poorly

1. **Lack of Inductive Biases**
   - CNNs like ResNet-18 leverage biases such as **spatial locality** and **translational invariance** to capture local patterns effectively.
   - ViTs, which rely on attention mechanisms and treat images as sequences of patches, lack these inductive biases.
   - **Impact**: ViTs struggle to generalize on smaller datasets like CIFAR-10, resulting in stagnation in accuracy and loss reduction.
2. **Dependence on Large-Scale Data**
   - ViTs are designed for large datasets like ImageNet-21k or JFT-300M, while CIFAR-10 (50,000 training images) is relatively small.
   - **Impact**: Insufficient data prevents ViT from learning robust representations, leading to slower learning and poor optimization.
3. **Model Complexity**
   - ViTs have higher parameter counts and quadratic complexity in their self-attention mechanisms, making them computationally demanding.
   - **Impact**: Prone to overfitting and inefficiencies in small datasets or resource-constrained environments.
4. **Limited Regularization**
   - CNNs benefit from inherent regularization due to structured kernels and pooling layers.
   - ViTs require external regularization methods such as dropout or advanced data augmentation, which were not extensively applied here.
   - **Impact**: Lack of aggressive regularization contributed to ViT's stagnation in training performance.
5. **Training Dynamics**
   - ViTs often need advanced learning rate schedules (e.g., warm-up or cosine annealing) for effective training.
   - **Impact**: Suboptimal training schedules hindered ViT's ability to converge effectively.

6.6 Observations on ResNet-18

Strengths

1. **Local Feature Extraction**:
   - Convolutional layers efficiently capture patterns (e.g., edges, textures), which are essential for CIFAR-10.

2. **Efficiency**:
    - Achieves high accuracy with low computational demands, making it ideal for resource-constrained environments.
3. **Regularization**:
    - Built-in inductive biases and pooling mechanisms naturally prevent overfitting.

Weaknesses

- Limited versatility for unstructured datasets where global attention mechanisms (e.g., ViTs) might be advantageous.

6.7 Observations on Vision Transformers (ViT)

Strengths

1. **Scalable Architecture**:
    - Designed for large-scale datasets, ViTs excel in tasks requiring global context and long-range dependencies.
2. **Versatility**:
    - When pretrained on large datasets, ViTs achieve state-of-the-art performance in image classification tasks.

Weaknesses

1. **Data Dependency**:
    - Require extensive pretraining on large datasets to perform well. Without inductive biases, ViTs struggle to extract meaningful features from small datasets like CIFAR-10.
2. **Overparameterization**:
    - ViTs have a high parameter count, making them prone to overfitting on small datasets and inefficient in resource-constrained settings.

6.8 Recommendations for Future Work

1. **Pretraining**:
    - Fine-tune a **pretrained ViT model** (e.g., pretrained on ImageNet) to leverage learned representations.
2. **Data Augmentation**:

- Use advanced techniques like **CutMix**, **Mixup**, or **RandAugment** to increase dataset size and diversity.

3. **Regularization**:
    - Introduce stronger regularization methods, such as dropout or label smoothing, to reduce overfitting.
4. **Learning Rate Optimization**:
    - Employ advanced learning rate schedules, such as warm-up or cosine annealing, to improve training dynamics.
5. **Alternative Architectures**:
    - Explore lightweight transformer variants like **MobileViT** or **DeiT** for smaller datasets and lower-resource environments.

7. Conclusion

This study compared the performance of Vision Transformer (ViT) and ResNet-18 on the CIFAR-10 dataset, focusing on metrics such as accuracy, efficiency, resource usage, and training dynamics. The results highlight a stark contrast between the two models in terms of their suitability for small-scale datasets.

ResNet-18 outperformed ViT across all performance metrics, achieving a near-optimal accuracy of **90.37%**, rapid convergence in training and validation losses, and computational efficiency with relatively stable epoch times. These strengths are attributed to ResNet-18's convolutional architecture, which effectively leverages inductive biases such as spatial locality and hierarchical feature extraction, making it well-suited for smaller datasets like CIFAR-10. ResNet-18's ability to generalize effectively demonstrates the robustness of its design in resource-constrained environments.

In contrast, ViT struggled to learn from the limited data, with accuracy stagnating at **27.90%** and both training and validation losses remaining high throughout the training process. The lack of inductive biases, dependency on large-scale pretraining, and computationally expensive self-attention mechanism rendered ViT less effective in this setting. Furthermore, the limited regularization and training optimizations applied in this study hindered ViT's performance, emphasizing the model's dependency on advanced configurations and larger datasets to unlock its full potential.

The resource usage analysis revealed that both models utilized similar GPU memory; however, ResNet-18 demonstrated higher computational efficiency by maintaining shorter epoch durations compared to ViT. This efficiency underscores the practical advantages of convolutional networks in environments where computational resources and dataset sizes are limited.

7.1 Key Findings

- **ResNet-18**:
  - High accuracy and rapid convergence.
  - Effective generalization and computational efficiency.
  - Suitable for resource-constrained environments and small datasets.
- **ViT**:
  - Poor performance on CIFAR-10 due to a lack of inductive biases.
  - Dependency on large-scale data and advanced training optimizations.
  - Higher computational demand with limited benefits on smaller datasets.

7.2 Implications

This study reaffirms the effectiveness of convolutional architectures like ResNet-18 for small-scale datasets. While ViTs offer state-of-the-art performance on large datasets with extensive pretraining, their adoption for smaller datasets and resource-constrained settings requires additional techniques such as pretraining, data augmentation, and advanced regularization methods.

7.3 Future Directions

1. **Pretraining**: Fine-tuning ViTs pretrained on larger datasets (e.g., ImageNet) may bridge the performance gap observed in this study.
2. **Data Augmentation**: Techniques like CutMix, Mixup, or RandAugment can help overcome data limitations for ViT.
3. **Optimized Architectures**: Exploring lightweight transformer variants, such as MobileViT or DeiT, could improve the applicability of ViTs for small datasets.
4. **Learning Rate Schedules**: Implementing advanced schedules, including warm-up and cosine annealing, can enhance ViT's training efficiency.

In conclusion, while ResNet-18 is well-suited for small-scale datasets like CIFAR-10, Vision Transformers require significant adaptations and additional resources to achieve comparable performance. Future research can address these limitations by leveraging transfer learning, enhanced regularization techniques, and tailored architectural adjustments to unlock the full potential of transformer-based models in constrained environments.

References

Maurício, J., Domingues, I., & Bernardino, J. (2023). Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review. *Applied Sciences, 13*(9), 5521. https://doi.org/10.3390/app13095521

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In Proceedings of the International Conference on Learning Representations (ICLR). Google Research, Brain Team. https://arxiv.org/abs/2010.11929

Appendix

Python code for the Experiment

```
#| comment: true
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from timm.models.vision_transformer import vit_base_patch16_224
from torchvision.models import resnet18
import time
import matplotlib.pyplot as plt
import json

# Hyperparameters
batch_size = 32  # Reduced batch size for lower VRAM usage
learning_rate = 0.001
num_epochs = 10
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Data Loading with Optimized Settings
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize for ViT compatibility
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize
])

train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
```

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

train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
                          shuffle=True, pin_memory=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
                         shuffle=False, pin_memory=True)

# Define Models
class VisionTransformerModel(nn.Module):
    def __init__(self, num_classes=10):
        super(VisionTransformerModel, self).__init__()
        self.model = vit_base_patch16_224(pretrained=True)
        self.model.head = nn.Linear(self.model.head.in_features, num_classes)

    def forward(self, x):
        return self.model(x)

class ResNet18Model(nn.Module):
    def __init__(self, num_classes=10):
        super(ResNet18Model, self).__init__()
        self.model = resnet18(pretrained=True)
        self.model.fc = nn.Linear(self.model.fc.in_features, num_classes)

    def forward(self, x):
        return self.model(x)

# Training Function with Mixed Precision and Memory Tracking
def train_model(model, train_loader, criterion, optimizer, scaler, device):
    model.train()
    total_loss = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        with torch.cuda.amp.autocast():
            outputs = model(images)
            loss = criterion(outputs, labels)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
```

```
        total_loss += loss.item()

        # GPU memory usage
        print(f"Memory Allocated:
              {torch.cuda.memory_allocated() / 1024 ** 2:.2f} MB")
        print(f"Memory Reserved:
              {torch.cuda.memory_reserved() / 1024 ** 2:.2f} MB")

    return total_loss / len(train_loader)

# Evaluation Function with Memory Tracking
def evaluate_model(model, test_loader, criterion, device):
    model.eval()
    correct = 0
    total = 0
    total_loss = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            with torch.cuda.amp.autocast():
                outputs = model(images)
                loss = criterion(outputs, labels)
            total_loss += loss.item()

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

            # GPU memory usage
            print(f"Memory Allocated:
                  {torch.cuda.memory_allocated() / 1024 ** 2:.2f} MB")
            print(f"Memory Reserved:
                  {torch.cuda.memory_reserved() / 1024 ** 2:.2f} MB")

    accuracy = 100 * correct / total
    return total_loss / len(test_loader), accuracy

# Initialize Models and Optimizers
vit_model = VisionTransformerModel(num_classes=10).to(device)
resnet_model = ResNet18Model(num_classes=10).to(device)

criterion = nn.CrossEntropyLoss()
vit_optimizer = torch.optim.Adam(vit_model.parameters(), lr=learning_rate)
resnet_optimizer = torch.optim.Adam(resnet_model.parameters(),
```

```
                                    lr=learning_rate)

scaler = torch.cuda.amp.GradScaler()  # For mixed precision

# Initialize Logs
vit_training_logs = {'loss': [], 'accuracy': []}
resnet_training_logs = {'loss': [], 'accuracy': []}

# Training and Evaluation
for epoch in range(num_epochs):
    print(f"Epoch {epoch+1}/{num_epochs}")

    start_time = time.time()

    # Train Vision Transformer
    vit_loss = train_model(vit_model, train_loader,
                           criterion, vit_optimizer, scaler, device)
    vit_val_loss, vit_accuracy = evaluate_model(vit_model,
                                                test_loader,
                                                criterion, device)

    # Train ResNet-18
    resnet_loss = train_model(resnet_model, train_loader,
                              criterion, resnet_optimizer, scaler, device)
    resnet_val_loss, resnet_accuracy = evaluate_model(resnet_model,
                                                      test_loader,
                                                      criterion, device)

    elapsed_time = time.time() - start_time

    # Append results to logs
    vit_training_logs['loss'].append(vit_loss)
    vit_training_logs['accuracy'].append(vit_accuracy)
    resnet_training_logs['loss'].append(resnet_loss)
    resnet_training_logs['accuracy'].append(resnet_accuracy)

    print(f"ViT - Loss: {vit_loss:.4f}, Val Loss:
          {vit_val_loss:.4f}, Accuracy: {vit_accuracy:.2f}%")
    print(f"ResNet18 - Loss: {resnet_loss:.4f}, Val Loss:
          {resnet_val_loss:.4f}, Accuracy: {resnet_accuracy:.2f}%")
    print(f"Epoch Time: {elapsed_time:.2f} seconds")

# Save Logs to JSON
with open("vit_training_logs.json", "w") as f:
```

```
    json.dump(vit_training_logs, f)
with open("resnet_training_logs.json", "w") as f:
    json.dump(resnet_training_logs, f)

# Visualization
# Accuracy Plot
plt.figure(figsize=(10, 5))
plt.plot(vit_training_logs['accuracy'], label='ViT Accuracy')
plt.plot(resnet_training_logs['accuracy'], label='ResNet18 Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Model Accuracy per Epoch')
plt.legend()
plt.show()

# Loss Plot
plt.figure(figsize=(10, 5))
plt.plot(vit_training_logs['loss'], label='ViT Loss')
plt.plot(resnet_training_logs['loss'], label='ResNet18 Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model Loss per Epoch')
plt.legend()
plt.show()

print("Training Complete!")
```