Report Title: **Pre-trained Models on ImageNet**

Author Name: **王小慧**

Student ID: **21307110415**

Institute: **信息科学与工程学院**

Report Date: **2024/12/6**

Contents

1 Introduction

In recent years, pretrained models have become an integral part of deep learning, enabling faster convergence and improved performance on various downstream tasks. The ability to fine-tune these models on task-specific data has led to significant advancements, particularly in large language models. There are several ways to fine-tune a model, with the most common approaches being **Full Fine-tuning (FFT)** and **Parameter Efficient Fine-tuning (PEFT)**.

1.1 Full Fine-tuning (FFT)

Full Fine-tuning (FFT) refers to the process of adjusting all the parameters of a pretrained model during the training phase on a new task. FFT typically results in better performance, as the model is free to adjust all of its weights to the task-specific data. However, this approach can be computationally expensive, especially for large models.

1.2 Parameter Efficient Fine-tuning (PEFT)

Parameter Efficient Fine-tuning (PEFT) focuses on making more targeted adjustments to the pretrained model, such that only a subset of the parameters are updated during the fine-tuning process. This approach aims to reduce the computational and memory overhead associated with full fine-tuning while still achieving competitive performance on downstream tasks.

1.2.1 Freezing Parameters

PEFT methods typically freeze most of the model' s parameters and only allow a limited number of parameters to be fine-tuned, such as the final classification layer or specific layers of interest. This can significantly reduce the number of trainable parameters and make fine-tuning more efficient.

1.2.2 Low-Rank Adaptation (LoRA)

One prominent technique in PEFT is Low-Rank Adaptation[1]. LoRA inserts low-rank matrices into specific parts of the model, allowing fine-tuning to focus on these smaller sets of parameters. The reparametrization of LoRA is shown in Figure 1. By applying low-rank updates, LoRA can effectively adapt the model to a new task with much less computational effort than traditional full fine-tuning. This results in a more

efficient use of resources and is particularly beneficial when working with large language models.

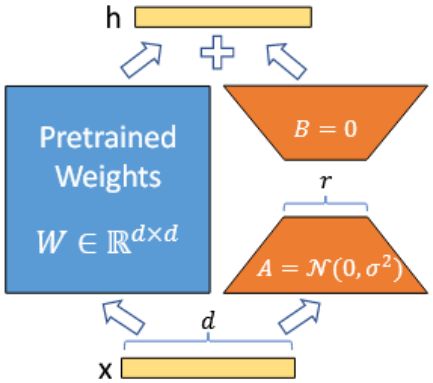

Figure 1 : The reparametrization of LoRA, it only trains A and B.

1.3 Vision Transformers (ViT)

Vision Transformers[2] represent a shift from traditional convolution-based architectures. Instead of applying convolutions, ViT treats an image as a sequence of patches, which are linearly embedded and processed by a transformer model. The ViT architecture is particularly known for its ability to capture long-range dependencies across the image, making it suitable for tasks requiring contextual understanding. The overall structure of ViT is presented in Figure 2.

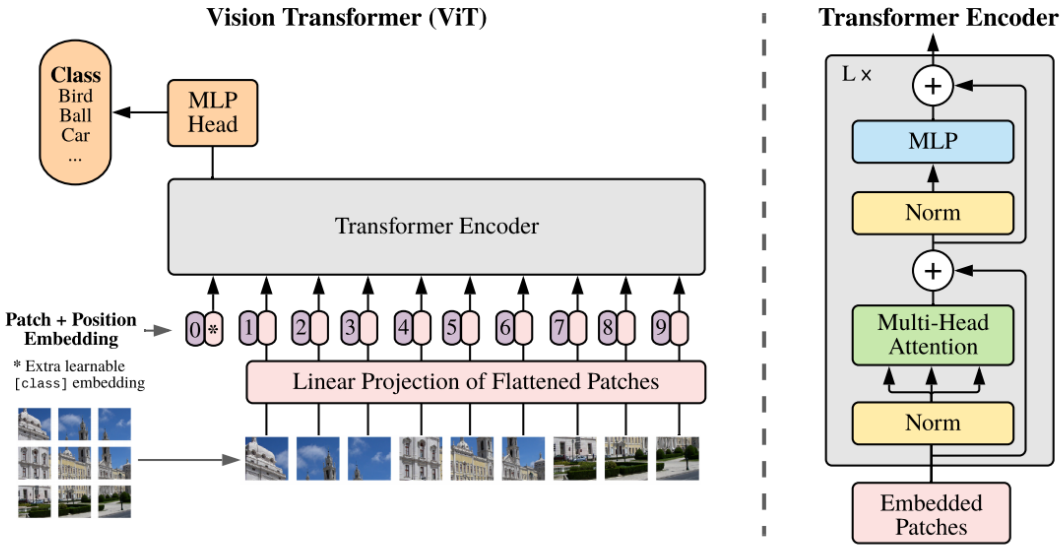

Figure 2 The Overall Structure of ViT

2 Experiment Setup

In this experiment, we explore the effectiveness of pretrained models, specifically ResNet, VGG, and Vision Transformers (ViT). These models are all pretrained on ImageNet. We aim to examine how fine-tuning pretrained models can enhance performance and efficiency in a downstream task, specifically on the CIFAR-10 dataset.

2.1 Model Preparation

Before the formal experiment, we first familiarize ourselves with the official structure of these models, so that we can effectively examine the behavior of different layers during the fine-tuning process. We use **torchvision.models** to load these three models.

2.1.1 ResNet18

A simplified version of the ResNet18 structure:

```
conv1: Conv2d (3 → 64)
bn1: BatchNorm2d (64)
relu: ReLU
maxpool: MaxPool2d
layer1: Sequential (BasicBlock x 2)
layer2: Sequential (BasicBlock x 2)
layer3: Sequential (BasicBlock x 2)
layer4: Sequential (BasicBlock x 2)
avgpool: AdaptiveAvgPool2d
fc: Linear (512 -> 1000)
```

2.1.2 VGG16

A simplified version of the VGG16 structure:

```
features: Sequential (Conv2d + ReLU x 13, MaxPool2d x 5)
avgpool: AdaptiveAvgPool2d
classifier: Sequential (Linear -> ReLU -> Dropout x 2 -> Linear -> ReLU ->
    Dropout -> Linear)
```

2.1.3 ViT_B_16

A simplified version of the ViT_B_16 structure:

```
conv_proj: Conv2d (3 -> 768, kernel_size=(16, 16), stride=(16, 16))
encoder: Encoder (12 layers)
  Each EncoderLayer:
    LayerNorm
    MultiheadAttention
```

```
        MLPBlock (Linear -> GELU -> Linear)
    ln: LayerNorm
    heads: Linear (768 -> 1000)
```

3 Experiment Procedure

3.1 ResNet18 without Pretrained Weights

In this section, we trained ResNet18 in **torchvision** from scratch to evaluate its performance compared to the previous ResNet18 in **Experiment 5**, as there might have been some changes. The results revealed that modifying the first convolutional layer (conv1) and the final fully connected layer (fc) to match the CIFAR-10 dataset is crucial for achieving good performance. Without these modifications, the model only achieved **30%** accuracy. However, after adjusting the conv1 and fc layers, the model reached an accuracy of **91.5%**, which is consistent with the results from Experiment 5.

```
model = models.resnet18(pretrained=True)
model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
model.maxpool = nn.Identity()  # move the maxpool layer
model.fc = nn.Linear(model.fc.in_features, 10)
```

Another approach is to resize the CIFAR-10 images to **224x224**, which is the input size of ImageNet. However, training with this resized input results in massive running time. We further examine the impact of resizing while full fine-tuning pre-trained models in subsection 3.4.

3.2 ResNet18 with Different Trainable Layers

In this section, we explore the impact of freezing certain layers while using pretrained weights, with the goal of understanding the influence of each layer in fine-tuning. The following experiments were conducted:

- Modify the **output layer (fc)** to be compatible with CIFAR-10 while keeping the other layers frozen.
- Modify both the **output layer (fc)** and the **first convolutional layer (conv1)** to fit CIFAR-10, freezing the remaining layers. **These changes to the output and conv1 layers are maintained throughout subsequent experiments.**
- Make a single layer trainable: either **Layer1, Layer2, Layer3 or Layer4**, while freezing the others.

- Make two layers trainable: combinations of **Layer1+Layer4, Layer1+Layer2, Layer2+Layer3, and Layer3+Layer4**, while freezing the rest of the layers.

```
    # Modify fc
    for param in model.parameters():
        param.requires_grad = False
    num_ftrs = model.fc.in_features
    model.fc = nn.Linear(num_ftrs, 10)
# ------------------------------------------------
    # Modify fc and conv1
    for param in model.parameters():
        param.requires_grad = False
    model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
    model.maxpool = nn.Identity()
    model.fc = nn.Linear(model.fc.in_features, 10)
# ------------------------------------------------
    # Make a single layer trainable
     for param in model.parameters():
        param.requires_grad = False
    ''' Maintain the changes to fc and conv1 '''
    layers_to_train = [model.layer1]
    for param in layer.parameters():
        param.requires_grad = True
# ------------------------------------------------
    # Make two layers trainable
    for param in model.parameters():
        param.requires_grad = False
    ''' Maintain the changes to fc and conv1 '''
    layers_to_train = [model.layer1, model.layer2]
    for layer in layers_to_train:
        for param in layer.parameters():
            param.requires_grad = True
```

3.3 Full Fine-tuning on ResNet18

In this section, we conduct a full fine-tuning experiment on the ResNet18 model. Unlike transfer learning approaches where only specific layers are updated, in full fine-tuning, we allow all layers to be trained. This enables the model to adapt more effectively to the CIFAR-10 dataset, leveraging the pre-trained knowledge from ImageNet while simultaneously learning features specific to CIFAR-10. To achieve efficient training, we modify the fully connected (fc) layers and the first convolutional layer (conv1) to better fit the CIFAR-10 dataset.

We also conducted an experiment using different learning rates for different layers. In full fine-tuning, it is important to apply different learning rates for different layers. Since

some layers already contain useful knowledge learned from the pre-trained task, they are fine-tuned with a smaller learning rate. On the other hand, the newly added layers, such as fc and conv1, need to learn new features specific to the CIFAR-10 dataset. Therefore, these layers require a higher learning rate to adapt quickly to the new task.

```
    optimizer = torch.optim.SGD([
        {'params': model.conv1.parameters(), 'lr': 7e-3},
        {'params': model.fc.parameters(), 'lr': 1e-2},
        {'params': model.layer1.parameters(), 'lr': 5e-3},
        {'params': model.layer2.parameters(), 'lr': 5e-3},
        {'params': model.layer3.parameters(), 'lr': 5e-3},
        {'params': model.layer4.parameters(), 'lr': 5e-3}
    ], momentum=0.9)
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=20)
```

3.4 Resize CIFAR-10 to 224x224

We also resize the CIFAR-10 images to 224x224. Resizing the CIFAR-10 dataset allows us to directly apply these pre-trained models without modifying the network architecture and may help the models generalize better. This experiment will allow us to directly compare the performance of the model when trained on the resized CIFAR-10 images versus the original smaller ones.

```
    transform_train = transforms.Compose([
        transforms.Resize(224),
        transforms.RandomRotation(10),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
    ])
    transform_test = transforms.Compose([
        transforms.Resize(224),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
    ])
```

3.5 Other Pre-trained Models

In this section, we explore other pre-trained models, including VGG16 and ViT_B_16. These models provide different types of architectures that may exhibit diverse behaviors when applied to the CIFAR-10 dataset. By evaluating these models, we aim to uncover the most suitable approach for CIFAR-10, examining their strengths, weaknesses, and overall performance.

4 Results and Analysis

4.1 Impact of Different Layers

4.1.1 Modify FC

In this part, we modified the output layer to match CIFAR-10 classification task while keeping the other layers frozen. The training and testing losses are presented in Figure 3. The model achieved **78%** accuracy in the first epoch but it struggled to improve further and ultimately reached only **80.9%**. This indicates that merely fine-tuning the output layer is insufficient; but it is also essential to adapt the fully connected layer to the specific class distribution.

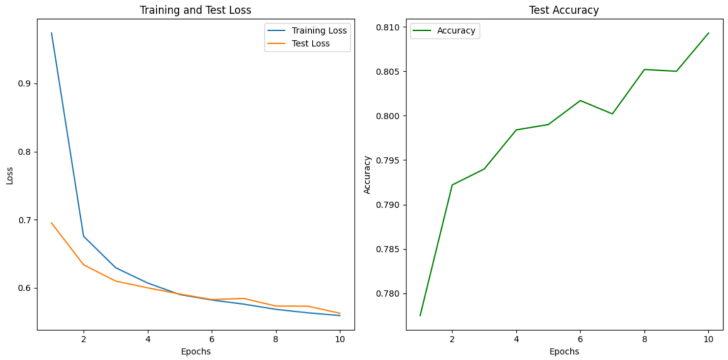

Figure 3 Training and Testing Losses when Modify FC

4.1.2 Modify FC and Conv1

In this part, we modified the first convolutional layer (conv1) to accommodate the CIFAR-10 input size of 32x32. The training and testing losses are shown in Figure 4. The result was only **68.2%**. We hypothesize that this performance drop occurred because the change to conv1 was significant, while the subsequent layers were frozen and unable to adapt to this adjustment.

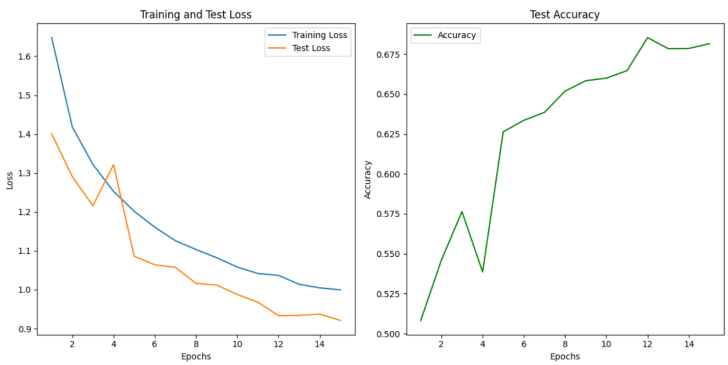

Figure 4 Training and Testing Losses when Modify FC and Conv1

4.1.3 Fine-tuning Different Layers

In this part, we further unfreeze some layers based on the poor performance observed in the previous experiment. We conducted several additional experiments, and the results are summarized in Table 1.

First, we unfroze Layer1 to Layer4 individually. Layer3 achieved the highest accuracy at **89.8%**, followed by Layer2 at **89.3%**, Layer4 at **89%**, and Layer1 at **88.4%**. While the results are quite similar, we believe the slight differences can be attributed to the fact that deeper layers capture higher-level, more abstract features, which are more critical for fine-tuning.

Next, we explored unfreezing two layers at a time. The results showed that the combinations of Layer1+Layer4 and Layer2+Layer3 outperformed Layer1+Layer2 and Layer3+Layer4. Specifically, Layer1+Layer4 and Layer2+Layer3 achieved much better results, with test accuracies of **92.7%** and **92.5%** respectively, compared to **89.3%** and **90.9%**. This suggests that it is more effective to spread the trainable layers across the network' s full structure, allowing the model to learn better hierarchical representations.

Finally, we performed full fine-tuning of the model, which led to the highest test accuracy of **93.7%**, surpassing the result obtained in Experiment 5 where we trained ResNet18 from scratch. This demonstrates the superiority of fine-tuning pretrained models for improving performance. Training and testing losses for this fine-tuning experiment are shown in Figure 5.

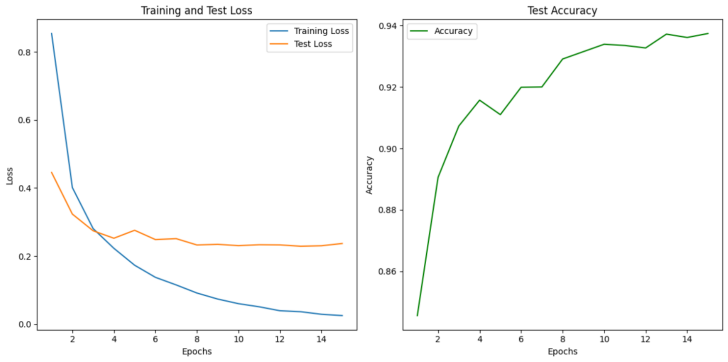

Figure 5 Full Fine-tuning on ResNet18

4.2 Impact of Different Learning Rate

In this part, we use different learning rates for different layers while performing full fine-tuning on ResNet18. Specifically, we assigned a lower learning rate for the layers which were already pre-trained, and a higher learning rate for the fully connected layer and the first convolutional layer (conv1), which were modified to suit the CIFAR-10 dataset. This approach allows the model to retain the learned representations in the

pre-trained layers while allowing the new layers to adapt more quickly to the new task.

The result of this experiment was a test accuracy of **94.1%**, which is higher than the result obtained with a fixed learning rate for all layers. This improvement suggests that by giving more flexibility to the newly initialized layers (fc and conv1) with a higher learning rate, the model is able to adapt more effectively to the specific characteristics of CIFAR-10. Training and testing losses are shown in Figure 6.

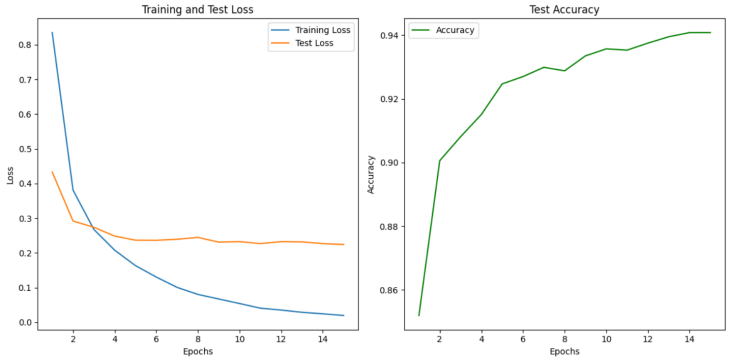

Figure 6 Full Fine-tuning with Different Learning Rate on ResNet18

4.3 Impact of Resize

In the previous part, we modified the first convolutional layer (conv1) to reduce training time and computational cost. In this part, we focused on maintaining the integrity of the entire pre-trained model by resizing CIFAR-10 images to 224x224. The result was remarkable, with the model achieving a test accuracy of **96.3%**. Training and testing losses are shown in Figure 7. This significant improvement can be attributed not only to preserving the full structure of the pre-trained model, which allows for better utilization of the learned features, but also to the higher image resolution.

However, the higher accuracy comes at a cost. Training on 224x224 images requires significantly more computational power and results in a considerably longer training time compared to training with the original 32x32 resolution.

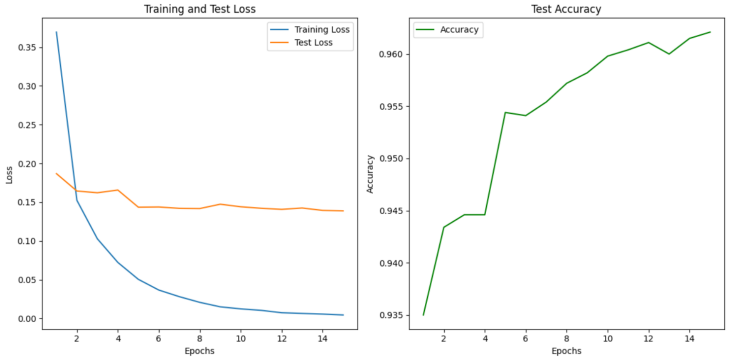

Figure 7 Resize CIFAR-10 to 224x224

4.4 Other Pre-trained Models

In addition to ResNet18, we also conducted experiments with other pre-trained models, specifically VGG16 and ViT_B_16. Unlike ResNet18, which adapted relatively well to the CIFAR-10 input size with minor adjustments, both VGG16 and ViT_B_16 posed more significant challenges in fitting the data without extensive changes to their network structures. For VGG16, we conducted experiments using both the original CIFAR-10 input size of 32x32 and the resized input size of 224x224. The results showed a noticeable difference in performance: with the 32x32 input size, VGG16 achieved an accuracy of **90.5%**, while resizing the input to 224x224 improved the performance to **95.1%**.

As a result, we focused primarily on resizing the CIFAR-10 images to a larger 224x224 resolution for ViT_B_16. Despite the additional computational cost, ViT_B_16 achieved the highest accuracy among all the experiments, with a test accuracy of **98.4%**. Training and testing losses are shown in Figure 8. The transformer-based architecture does not rely on convolutions, allowing it to process spatial relationships in a more flexible and efficient manner.

While ViT' s outstanding accuracy is a testament to the strength of transformer models, it also came with a trade-off. The higher accuracy achieved by ViT required significantly more computational resources and longer training time than both ResNet and VGG. This highlights a common trade-off when using more complex models and higher resolutions.

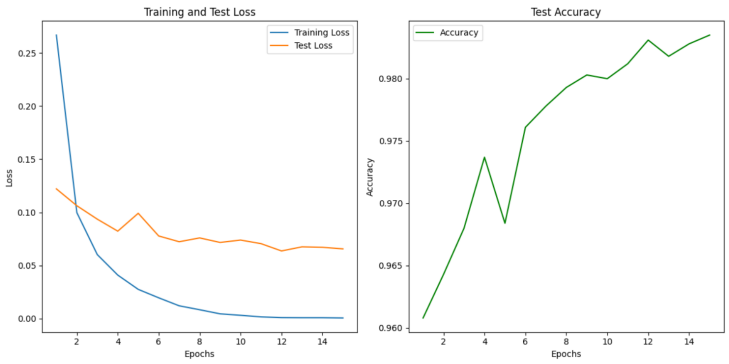

Figure 8 Full Fine-tuning on ViT

5 Conclusion

In this series of experiments, we first investigated the impact of training different layers of ResNet18 by freezing specific layers and examining their effects on performance. Initially, training only the final output layer (fc) resulted in a moderate accuracy of **80.9%**. When we included the first convolutional layer (conv1) in the training process,

performance significantly dropped. We further examined other layers like Layer1 and Layer4. The best results, achieving **93.7%** accuracy, were obtained when all layers were fine-tuned.

Next, we experimented with resizing the CIFAR-10 dataset from its original 32x32 resolution to 224x224. This change led to a significant improvement in accuracy, with the model achieving **96.3%**. However, this improvement came at the cost of increased computational time and resource demands, which is an important trade-off when considering practical applications.

Additionally, we extended our experiments to other pre-trained models, including VGG16 and Vision Transformer (ViT_B_16). The performance of ViT_B_16 outperformed both ResNet18 and VGG16, achieving the highest accuracy across all experiments, which is **98.4%**, underscoring the potential of Vision Transformers in handling image classification tasks.

In summary, the results of our experiments suggest that fine-tuning can significantly improve the performance of models. Models like ViT demonstrated superior performance, suggesting that exploring alternative architectures could yield even better outcomes for image classification tasks.

Table 1 Experiment Results

Model	Input Size	Trainable Layers	Test Acc
ResNet18	32x32	fc	80.9
ResNet18	32x32	fc, conv1	68.2
ResNet18	32x32	fc, conv1, layer1	88.4
ResNet18	32x32	fc, conv1, layer2	89.3
ResNet18	32x32	fc, conv1, layer3	89.8
ResNet18	32x32	fc, conv1, layer4	89.0
ResNet18	32x32	fc, conv1, layer1, layer4	92.7
ResNet18	32x32	fc, conv1, layer2, layer3	92.5
ResNet18	32x32	fc, conv1, layer1, layer2	89.3
ResNet18	32x32	fc, conv1, layer3, layer4	90.9
ResNet18	32x32	FFT	93.7
ResNet18	32x32	FFT with different lr	94.1
ResNet18	224x224	FFT	**96.3**
VGG16	32x32	FFT	90.5
VGG16	224x224	FFT	95.1
ViT_B_16	224x224	FFT	**98.4**

References

[1] J. Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685, 2021.

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.