Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprisingly Well on ImageNet

Luke Melas-Kyriazi
Oxford University
lukemk@robots.ox.ac.uk

Abstract

The strong performance of vision transformers on image classification and other vision tasks is often attributed to the design of their multi-head attention layers. However, the extent to which attention is responsible for this strong performance remains unclear. In this short report, we ask: is the attention layer even necessary? Specifically, we replace the attention layer in a vision transformer with a feed-forward layer applied over the patch dimension. The resulting architecture is simply a series of feed-forward layers applied over the patch and feature dimensions in an alternating fashion. In experiments on ImageNet, this architecture performs surprisingly well: a ViT/DeiT-base-sized model obtains 74.9% top-1 accuracy, compared to 77.9% and 79.9% for ViT and DeiT respectively. These results indicate that aspects of vision transformers other than attention, such as the patch embedding, may be more responsible for their strong performance than previously thought. We hope these results prompt the community to spend more time trying to understand why our current models are as effective as they are.[1]

1. Introduction

Introduced by [4], the vision transformer architecture applies a series of transformer blocks to a sequence of image patches. Each block consists of a multi-head attention layer [11] followed by a feed-forward layer (i.e. a linear layer, or a single-layer MLP) applied along the feature dimension. The general-purpose nature of this architecture, coupled with its strong performance on image classification benchmarks, has prompted significant interest from the vision community. However, it is still not clear exactly *why* the vision transformer is effective.

The most-cited reason for the transformer's success on vision tasks is the design of its attention layer, which gives the model a global receptive field. This layer may be seen as a data-dependent linear layer, and when applied on image patches it resembles (but is not exactly equivalent to) a convolution. Indeed, a significant amount of recent work has gone into improving the efficiency and efficacy of the attention layer.

In this short report, we conduct an experiment that hopes to shed a little light on why the vision transformer works so well in the first place. Specifically, we remove attention from the vision transformer, replacing it with a feed-forward layer applied over the patch dimension. After this

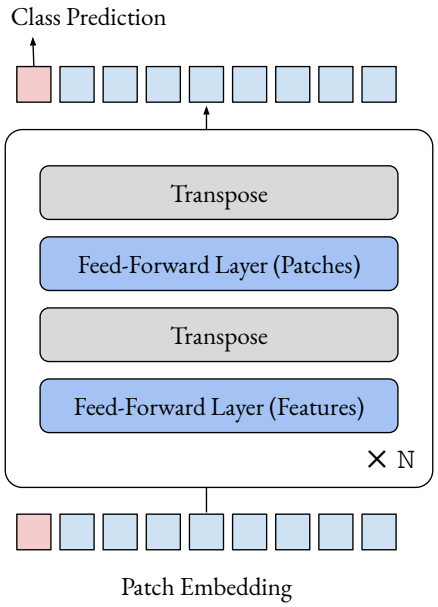

Figure 1: The architecture explored in this report is extremely simple, consisting of a patch embedding followed by a series of feed-forward layers. These feed-forward layers are alterately applied to the patch and feature dimensions of the image tokens. The architecture is identical to that of ViT [4] with the attention layer replaced by a feed-forward layer.

[1] Code is available at https://github.com/lukemelas/do-you-even-need-attention

change, the model is simply a series of feed-forward layers applied over the patch and feature dimensions in an alternating fashion (Figure 1).

In experiments on ImageNet (Table 1), we show that quite strong performance is attainable even without attention. Notably, a ViT-base-sized model gives 74.9% top-1 accuracy without any hyperparameter tuning (i.e. using the same hyperparameters as its ViT counterpart). These results suggest that the strong performance of vision transformers may be attributable less to the attention mechanism and more to other factors, such as the inducive bias produced by the patch embedding and the carefully-curated set of training augmentations.

The primary purpose of this report is to explore the limits of simple architectures. We do not to break the ImageNet benchmarks; on that front, methods such as neural architecture search (e.g. EfficientNet [8]) will inevitably perform best. Nonetheless, we hope that the community finds these results interesting, and that these results prompt more researchers to investigate why our current models are as effective as they are.

2. Background

The context for this report is that over the past few months, there has been an explosion of research into variants of the vision transformer architecture: Deit [9] adds distillation, DeepViT [14] mixes the attention heads, CaiT [10] separates the attention layers into two stages, Token-to-Token ViT [13] aggregates neighboring tokens throughout the network, CrossViT [1] processes patches at two scales, PiT [6] adds pooling layers, LeViT [5] uses convolutional embeddings and modified attention/normalization layers, CvT [12] uses depthwise convolutions in the attention layer, and Swin/Twins [2, 7] combines global and local attention, just to name a few.

These works improve upon the vision transformer architecture, each showing strong performance on ImageNet. However, it is not clear how the different parts of ViT or its many variants contribute to the final performance of each of these models. This report details an experiment that investigates one aspect of this issue, namely how important the attention layers are to ViT's success.

3. Method and Experiments

3.1. Do You Even Need Attention?

We remove the attention layer from the ViT model, replacing it by a simple feed-forward layer over the patch dimension. We use the same structure as the standard feed-forward network over the feature dimension, which is to say that we project the patch dimension into a higher-dimensional space, apply a nonlinearity, and project it back to the original space. Figure 2 gives PyTorch code for a single block of the feed-forward-only transformer.

		Params	ImageNet Top-1
Tiny ($P = 16$)	ViT	-	-
	DeiT	5.7M	72.2
	FF Only	7.7M	61.4
Base ($P = 16$)	ViT	86M	77.9
	DeiT	86M	79.9
	FF Only	62M	74.9
Large ($P = 32$)	ViT	306M	71.2
	DeiT	-	-
	FF Only	206M	71.4

Table 1: A comparison of ImageNet top-1 accuracies for different model sizes. In the first column, P refers to the patch size in pixels. Overall, the models with only feed-forward layers (*FF Only*) perform worse than their counterparts with attention, but they perform surprisingly well regardless. Performance deteriorates for the largest models both with and without attention.

We should note that, like the vision transformer and its many variants, this feed-forward-only network bares strong resemblance to a convolutional network. In fact, the feed-forward layer over the patch dimension can be viewed as an unusual type of convolution with a full receptive field and a single channel. Since the feed-forward layer over the feature dimension can be seen as a 1x1 convolution, it would be technically accurate to say that the entire network is a sort of convolutional network in disguise. That being said, it is certainly closer to a transformer than to a traditionally-designed convolutional network (e.g. ResNet/VGG).

3.2. Experimental Setup

We train three models, corresponding to the ViT/DeiT tiny, base, and large networks, on ImageNet [3] using the setup from DeiT [9]. The tiny and base networks have patch size 16, while the large network has patch size 32 due to computational constraints. Training and evaluation is performed at resolution 224px. Notably, we use exactly same hyperparameters as DeiT for all models, which means that our performance could likely be improved with hyperparameter tuning.

3.3. Results

Table 1 shows the performance of our simple feed-forward network on ImageNet. Most notable, the feed-forward-only version of ViT/Deit-base achieves surprisingly strong performance (74.9% top-1 accuracy), comparable to a number of older convolutional networks (e.g. VGG-16, ResNet-34). Such a comparison is not exactly fair because the feed-forward model uses stronger training aug-

mentations, but it is nevertheless quite a strong result in an absolute sense.

Performance deteriorates for the large models for the models both with and without attention, yeilding 71.2% and 71.4% top-1 accuracy respectively. As detailed in [4], pre-training with a larger dataset appears necessary for such enormous models.

3.4. Do You Even Need Feed-Forward Layers?

Naturally, since we tried training a model with only feed-forward layers, we also tried training a model with only attention layers. In this model, we simply replaced the feed-forward layer over the feature dimension with an attention layer over the feature dimension. We only experimented with a tiny-sized model (4.0M parameters), but it performed spectacularly poorly (28.2% top-1 accuracy at 100 epochs, at which point we ended the run).

3.5. Discussion

The above experiments demonstrate that it is possible to train a reasonably strong transformer-style image classifiers without attention layers. Furthermore, attention layers without feed-forward layers do not appear to yield similarly strong performance. These results indicate that the strong performance of ViT may be attributable more to its patch embeddings and training procedure than to the design of the attention layer. The patch embeddings in particular provide a strong inductive bias that is likely one of, if not the, main drivers of the model's strong performance.

As mentioned above, the purpose of this report is to better understand vision transformers, not to develop new state-of-the-art architectures. From a practical perspective, the feed-forward-only model suffers from many of the same drawbacks as vision transformers. Both use $O(n^2)$ memory complexity with respect to the sequence length: the feed-forward-only models directly stores $O(n^2)$ parameters while the attention model constructs a $O(n^2)$-sized matrix during its forward pass. The feed-forward-only model also only works for fixed-length sequences due to the second feed-forward layer. This is not a big issue for image classificaion, where images are cropped to a standard size, but it limits their applicability to other tasks.

Despite their drawbacks, feed-forward-only models shed light on vision transformers and attention mechanisms in general. For the future, it would be interesting to investigate the extent to which these conclusions apply outside of the image domain, for example in NLP/audio.

3.6. Conclusion

This short report demonstrates that transformer-style networks without attention layers make for surprisingly strong image classifiers. Future work in this direction could attempt to better understand the contributions of other pieces of the transformer architecture (e.g. the normalization layer or initialization scheme). More broadly, we hope that this short report encourages further investigation into why our current models perform as well as they do.

References

[1] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification, 2021.

[2] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting spatial attention design in vision transformers, 2021.

[3] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255, 2009.

[4] 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, 2020.

[5] Ben Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference, 2021.

[6] Byeongho Heo, Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers, 2021.

[7] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021.

[8] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[9] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers and distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[10] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[12] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.

[13] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.

[14] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer, 2021.

```
from torch import nn

class LinearBlock(nn.Module):
  def __init__(self, dim, mlp_ratio=4., drop=0., drop_path=0., act=nn.GELU,
               norm=nn.LayerNorm, n_tokens=197):  # 197 = 16**2 + 1
    super().__init__()
    self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()

    # FF over features
    self.mlp1 = Mlp(in_features=dim, hidden_features=int(dim*mlp_ratio), act=act, drop=drop)
    self.norm1 = norm(dim)

    # FF over patches
    self.mlp2 = Mlp(in_features=n_tokens, hidden_features=int(n_tokens*mlp_ratio), act=act, drop=drop)
    self.norm2 = norm(n_tokens)

  def forward(self, x):
    x = x + self.drop_path(self.mlp1(self.norm1(x)))
    x = x.transpose(-2, -1)
    x = x + self.drop_path(self.mlp2(self.norm2(x)))
    x = x.transpose(-2, -1)
    return x


class Mlp(nn.Module):
  def __init__(self, in_features, hidden_features, act_layer=nn.GELU, drop=0.):
    super().__init__()
    self.fc1 = nn.Linear(in_features, hidden_features)
    self.act = act_layer()
    self.fc2 = nn.Linear(hidden_features, in_features)
    self.drop = nn.Dropout(drop)

  def forward(self, x):
    x = self.fc1(x)
    x = self.act(x)
    x = self.drop(x)
    x = self.fc2(x)
    x = self.drop(x)
    return x
```

Figure 2: PyTorch code for a single transformer block consisting of two feed-forward layers