Implementation of Less Attention ViT for Image Classification and Investigation on XGBoost classifier*

*For the fulfillment project proposal of AT82.05 Computer Vision course by Dr. Cherdsak Kingkan

Kaung Nyo Lwin
Department of Data Sciences and Artificial Intelligence
Institute of Technology
Pathum Thani, Thailand
st125066@ait.asia

Abstract—The project is to build an image classification model based on the vision transformer model by integrating the mechanism to reduce complexity of attention computation. Then, this model will be tested on a gradient boosting classifier.

I. INTRODUCTION

In recent years, the field of computer vision has advanced rapidly, largely due to improvements in deep learning techniques and the availability of large datasets. Among the most notable deep learning methods, Convolutional Neural Networks (CNNs) [8] have proven highly effective, achieving remarkable results in various tasks such as image classification [8, 28], object detection [5, 22], and semantic segmentation [1, 23].

Inspired by the success of Transformers [27] in natural language processing, Vision Transformers (ViTs) [4] treat images as a sequence of tokens. These tokens are encoded to create an attention matrix, which is a core component of the self-attention mechanism. However, the computational complexity of this self-attention mechanism increases quadratically with the number of tokens, and this becomes even more challenging when processing high-resolution images. To address this, some researchers have explored ways to reduce token redundancy through dynamic selection [9, 21] or token pruning [33], which can help alleviate the computational burden while maintaining performance comparable to standard ViTs. However, token reduction and pruning methods require careful design of the selection process and may inadvertently discard important tokens.

In a recently published paper[1000], an approach to modify the fundamental architecture of standard ViT by introducing the LessAttention Vision Transformer (LaViT) is proposed. Since it is a recently published in 2024 and source codes are not published, I will implement LaViT based on the standard ViT. Then, investigation on their proposed modifications and the combination of the gradient boosting classifier as the classifier of this implemented model.

II. RELATED WORKS

The Transformer architecture, originally developed for machine translation [27], has since been adapted for computer vision tasks with the advent of Vision Transformers (ViT) [4]. The primary innovation of ViT lies in its ability to capture long-range dependencies between distant regions of an image, achieved through the use of self-attention mechanisms. Building on the success of ViT, a variety of derivative models have been proposed, each designed to address specific limitations of the original architecture. For example, DeiT [25] improves data efficiency during training by introducing a distillation token. Meanwhile, CvT [32] and CeiT [36] incorporate convolutional structures into the ViT framework, combining the spatial invariance of CNNs with the long-range dependency modeling of ViTs. These developments highlight the continued evolution of transformer-based models in computer vision.

Although ViTs are highly effective, they come with significant computational costs. Research into more efficient vision transformers seeks to address the quadratic complexity of the self-attention operation by incorporating hierarchical downsampling [17, 29, 30], token reduction [9, 21, 33], or lightweight architectural designs [18, 19]. Hierarchical downsampling reduces the number of tokens progressively across stages, which helps mitigate the quadratic computation cost of self-attention while enabling ViTs to capture hierarchical structures in the data [17, 18, 29, 30]. Another approach focuses on token selection, where the least informative tokens are discarded to reduce computational load. For example, methods in [9, 21, 33] reorganize image tokens by retaining the most informative ones and discarding those that receive minimal attention, thus accelerating the following multi-head self-attention (MHSA) and feed-forward network (FFN) operations.

III. METHODOLOGY

There are four major modifications to be implemented to build LaViT based on the standard vision transformer as follows:

- Downsampling Operation
- Less-Attention Layer
- Residual-based Attention Downsampling
- Diagonality Preserving Loss

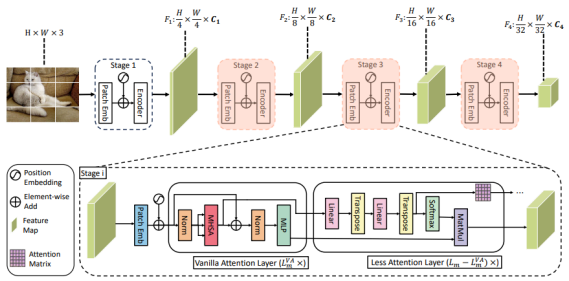

Fig. 1: Architecture of LaViT

Different from standard vision transformers, each stage of the transformer layer performs patch embedding. This can be seen in figure 1, the architecture of LaViT.

A. *Downsampling Operation*

Several studies have introduced hierarchical structures into ViTs, inspired by the success of hierarchical architectures in CNNs. These approaches divide the Transformer blocks into multiple stages and apply downsampling operations before each stage, effectively reducing the sequence length.

LaViT employs a convolutional layer with the kernel size and stride set to 2 to reduce the attention tokens in each stage of the transformer layers. This approach enables flexible scaling of the feature map at each stage, thereby creating a Transformer hierarchical structure that mirrors the organization of the human visual system.

B. *Less-Attention Layer*

After the standard multi-head self-attention (MHSA) operation to capture the overall long-range dependencies, the lees-attention layers are introduced in each stage of transformer layers. These layers mitigate quadratic computation and address attention saturation by applying a linear transformation to the stored attention scores.

It can be seen, in fig 1, that there are two linear layers that transform the attention matrix into the dimension of attention sequence length (N). These layers learn to rank the attention matrix with the softmax function so that the downsampling operation in each state does not remove the important information. The output matrices of these two layers are transposed in order to serve the purpose of maintaining the matrix similarity behavior.

Finally, the attention scores from the Less-Attention layer and the output from the standard MHSA are multiplied before feeding the attention to the next layers.

C. *Residual-based Attention Downsampling*

In hierarchical ViTs, downsampling operations are commonly applied to the feature map as the computation progresses through stages. While this technique helps reduce the number of tokens, it can also lead to the loss of important contextual information.

In order to mitigate this issue, LaViT uses a skip connection from the previous layer so that the attention from previous layers can guide the attention computation in the current layer. However, since the dimensions of attention matrices are different in each layer, a convolution layer to extract the information of the previous layers is employed. The dimension of filter and stride should be chosen to make the dimensions the same with those of the subsequent attention matrices.

D. *Diagonality Preserving Loss*

In the paper of LaViT, the loss function which is designed to mitigate losing the spatial information is introduced. As a conventional attention matrix should possess the properties of the diagonality and symmetry, the following equation is used as a part of the loss function.

$$\mathcal{L}_{\mathrm{DP},l} = \sum_{i=1}^{N}\sum_{j=1}^{N} |\mathbf{A}_{ij} - \mathbf{A}_{ji}| + \sum_{i=1}^{N}((N-1)\mathbf{A}_{ii} - \sum_{j\neq i}\mathbf{A}_{j}).$$

This diagonality loss is added to the cross entropy loss of the classification.

IV. IMPLEMENTATION

For implementation of LaViT, I base the standard vision transformer, shown in figure 2, with four transformer layers, 32 embedding dimensions and two heads of self -attention. This model is used for comparison as a baseline.

```
class ViT(nn.Module):
    def __init__(self, ch=3, img_size=32, patch_size=4, emb_dim=32,
                 n_layers=4, out_dim=10, dropout=0.1, heads=2):
        super(ViT, self).__init__()
```

Fig. 2: Standard ViT

A. Downsampling Operation

In order to implement the mechanism for downsampling, the patch embedding layer is modified as shown in figure 3. Since each patch of the image is encoded into the embedding dimension, the attention matrix needs to be rearranged into the dimension of the image so that the convolution layer can extract the important features. Then, the convolution filter with filter and stride set to two, as recommended in the paper, is used in the convocation layer. For simplicity and computational resource limitations, the number of filters is only set to one for each target of the transformer layers.

Although batch normalization and activation are not mentioned in the paper, I find that these layers improve the performance of the model. After downsampling operation, the extracted features are rearranged to perform embedding.

The downsampling is not performed in the initial stage of transformer layers so that the first layer can capture the full dependencies of the attention sequence.

```
self.projection = nn.Sequential(
    Rearrange('b (h w) e -> b e h w',h=n_patch,w=n_patch),
    torch.nn.Conv2d(in_channels,out_channels,2,2),
    nn.BatchNorm2d(out_channels),
    nn.LeakyReLU(0.1),
    Rearrange('b c h w -> b (h w) c'),
    nn.Linear(out_channels, emb_size)

)
```

Fig. 3: Attention downsampling operation

B. Less-Attention Layer

Then, the less-attention layer is implemented to transform the attention matrix. The dimension output dimension of this layer is the number of patches of the patch embedding of each stage. It can be seen in figure 4.

```
class LessAttention(nn.Module):
    def __init__(self,n_patch, emb_dim):
        super().__init__()
        self.in_dim = emb_dim
        self.n_patch = n_patch

        self.FC1 = torch.nn.Linear(self.in_dim, n_patch)
        self.FC2 = torch.nn.Linear(n_patch, n_patch)

    def forward(self, x):
        x = self.FC1(x)
        x =  torch.transpose(x, 1, 2)
        x = self.FC2(x)
        x =  torch.transpose(x, 1, 2)
        return x
```

Fig. 4: Less-Attention layer

C. Residual Attention

In order to add the residual connection between the consecutive layers, the residual-based downsampling is performed using a convolutional layer with the filter size four, to extract summarised information of the layer to be passed to the next attention computation. The in-channel and out-channel are set to one as mentioned in the paper. The filter size four is chosen because it makes the dimensions of the matrix matched with the next layer. The code snippet is shown in figure 4.

```
transformer_block = nn.Sequential(
    PatchEmbedding(in_channels=emb_dim,
                   out_channels=emb_dim,ConV=True,
                   n_patch=self.hw_patch,emb_size=emb_dim),
    ResidualAdd(PreNorm(emb_dim,
                        Attention(emb_dim, n_heads = heads,
                                  dropout = dropout))),
    ResidualAdd(PreNorm(emb_dim,
                        FeedForward(emb_dim, emb_dim,
                                    dropout = dropout))),
    PreNorm(emb_dim,LessAttention(num_patches,emb_dim)),
    Rearrange('b h w -> b 1 h w'),
    nn.Conv2d(1,1,2,4),
    Rearrange('b 1 h w -> b h w'),
    )
```

Fig. 5: Residual-Based Attention downsampling operation

Finally, I try to implement the diagonal preserving loss function, as shown in the figure 6. The first part of the loss is to preserve the symmetry by clarifying the values in inverse positions of the matrix should not be much different .

The second part is to preserve diagonality of the attention matrix by clarifying that the diagonals of the

attention matrices should be greater than others in the attention matrices. However, in training the model, these losses are dominant over classification loss, causing the model to suffer both low bias and low variance in terms of accuracy. Therefore, they are excluded in the final training.

```
for i in range(layers):
    b,n,_ =  att_matrix[i].shape
    if i == 0:
        continue
    self.loss1 += torch.sum(
        torch.abs(att_matrix[i] - torch.transpose(att_matrix[i], 1, 2)
                  )
    )
    dia = torch.diagonal(att_matrix[i][:,:,:],dim1=1,dim2=2)
    dia = dia * (n - 1)
    no_dai = att_matrix[i].clone()
    for j in range(b):
        no_dai[j] = no_dai[j].clone().fill_diagonal_(0)
    no_dai_sum = torch.sum(no_dai,(1,2)).reshape(-1,1)
    self.loss2 += torch.sum(torch.sub(dia,no_dai_sum))
```

Fig. 6: Diagonality preserving loss

V. EXPERIMENTS AND DISCUSSION

A. *Experimental Setting*

To evaluate the models, the accuracy score and training period are chosen as the main metrics because the tradeoff between computational complexity and performance of the model intend to be measured.

Due to the limitation of the computational resources and time frame, I choose two lightweight datasets, CIFAR-10 and MNIST. Furthermore, I subset these datasets into 1000 samples for training and each 200 and validation because I intend to find out how well these models perform on a limited range of samples.

Both standard ViT and LaViT are modeled with the same parameters as shown in figure 2 except that the patch size is set to 2 in LaViT in order to preserve the enough information in the last layer of the transformer encoder block.

For training, the learning rate of 0.001 and epochs of 100 and AdamW optimizer are set for both models.

In order to monitor how the performance of the model is improved over each epoch of the training, I choose to use the learning curve with accuracy score.

B. *Accuracy Evaluation*

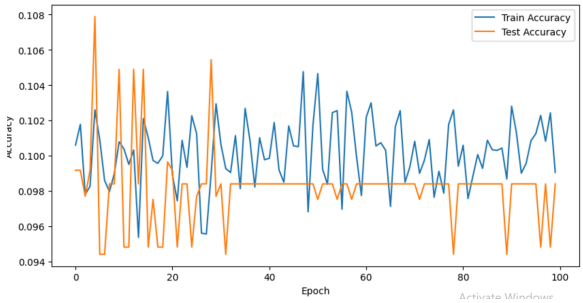

Fig. 7: Accuracy of standard ViT trained on CFAR-10

In figure 7, it can be seen that both training accuracy and testing accuracy of the model, trained on CFAR-10, are below 0.1 over 100 epochs. This shows that the model does not learn anything within 100 epochs with 1000 samples of data.

When I test the model, the model precincts the same class for all test samples, showing that the model does not learn the nuanced features of the images.

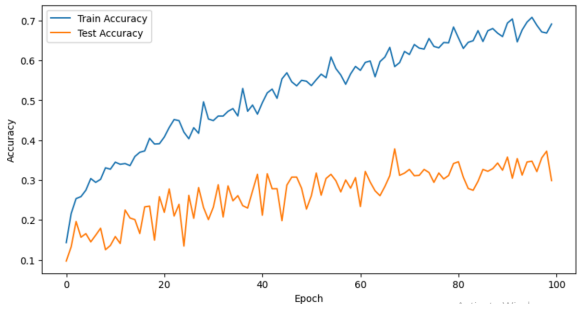

Fig. 8: Accuracy of LaViT trained on CFAR-10

On the other hand, both training accuracy and testing accuracy of LaViT, as shown in figure 8, are improving over each epoch. It can be observed that the model is learning well with 1000 samples of data.

Based on the some experiments, batch normalization nan Leaky RELU after in attention downsampling layer improves the performance of the model around 0.5 accuracy score. The average accuracy after 100 epoch training is 0.3.

The results show that attention downsampling operation is making the model more robust by extracting more necessary information for image classification.

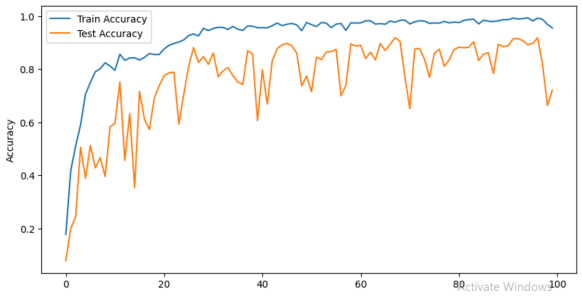

Fig. 9: Accuracy of LaViT trained on MNIST

The results of the training accuracy and testing accuracy are shown in figure 9. It is better than the CFAR-10 model. This may be because the MNIST data are more simple than CFAR-10.

The accuracy of training and testing are not much different over the whole training. I can conclude that LaViT is much more accurate than standard ViT. On the

C. *Training Time Evaluation*

	ViT	LaViT
CFAR-10	-	385.823026
MNIST	10451.67314	376.891415

Table 1: Training Time in seconds

The table 1 shows the training time in seconds of the standard ViT and LaViT. The standard model is extremely slow compared to the LaViT. While the LaViT model training is just around six minutes, the standard ViT took over two hours of training. Hence, The LaViT is much more efficient than the standard ViT in terms of computational complexity. This shows that the LaViT is extracting necessary features using attention downsampling operations, making the model robust and faster.

D. *Testing with Gradient Boosting Classifier*

After training the LaViT model, I tried to test the gradient boosting classifier on the extracted features. As the gradient boosting classifier requires two dimensional data, the embedded features are flattened and the model is trained on them. However, the results are not good, as shown in figure 10.

	precision	recall	f1-score	support
0	0.12	0.16	0.14	19
1	0.09	0.12	0.10	16
2	0.00	0.00	0.00	23
3	0.13	0.07	0.09	28
4	0.14	0.10	0.12	20
5	0.00	0.00	0.00	11
6	0.04	0.07	0.05	15
7	0.07	0.04	0.05	28
8	0.11	0.11	0.11	18
9	0.15	0.14	0.14	22
accuracy			0.08	200
macro avg	0.08	0.08	0.08	200
weighted avg	0.09	0.08	0.08	200

Fig. 10: Classification report Gradient Boosting Classifier model

This may be because the features are extracted to fit in the linear classier rather than gradient boosting classifier.

VI. CONCLUSION

In this project, I have implemented a LaViT model based on a recently published paper. According to the experimental results, the LaViT model is more efficient and more accurate than the standard ViT.Since the limitation of timeframe and computational resources, the experimental settings are designed to find out the results for a limited range of samples of the lightweight datasets and a limited model complexity.

The results of the experiments show that LaViT outperforms the standard ViT in both computational complexity and accuracy. Therefore, as the future work, the experiments will be expanded for border setting to see the general performance.

Moreover, the gradient boosting classifier can be tested by integrating it in training the LaViT model so that the model extracts the features that are fit for the classifier. This will be noted as future work.

VII. SOURCE CODE

```
class ViT(nn.Module):
    def __init__(self, ch=3, img_size=32, patch_size=2, emb_dim=32,
                 n_layers=4, out_dim=10, dropout=0.1, heads=2):
        super(ViT, self).__init__()
        self.channels = ch
        self.height = img_size
        self.width = img_size
        self.patch_size = patch_size
        self.n_layers = n_layers
        self.att_matrix = list(range(self.n_layers))
        self.attention_feature = None
        # Patching
        self.patch_embedding = PatchEmbedding(in_channels=ch,patch_size=patch_size, emb_size=emb_dim)
        num_patches = (img_size // patch_size) ** 2
        self.hw_patch = int(np.sqrt(num_patches))
        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, emb_dim))
        self.cls_token = nn.Parameter(torch.rand(1, 1, emb_dim))
        first_layer = True
        self.layers = nn.ModuleList([])
        for _ in range(n_layers):
            if first_layer:
                transformer_block = nn.Sequential( ResidualAdd(PreNorm(emb_dim, Attention(emb_dim, n_heads = heads, dropout = dropout))),
                    ResidualAdd(PreNorm(emb_dim, FeedForward(emb_dim, emb_dim, dropout = dropout)))
                    )
                first_layer = False
                self.layers.append(transformer_block)
                continue
            else:
                num_patches = int(num_patches / 4)
                transformer_block = nn.Sequential(PatchEmbedding(in_channels=emb_dim,out_channels=emb_dim,ConV=True,
                                n_patch=self.hw_patch,emb_size=emb_dim),
                    ResidualAdd(PreNorm(emb_dim, Attention(emb_dim, n_heads = heads, dropout = dropout))),
                    ResidualAdd(PreNorm(emb_dim, FeedForward(emb_dim, emb_dim, dropout = dropout))),
                    PreNorm(emb_dim,LessAttention(num_patches,emb_dim)),
                    Rearrange('b h w -> b 1 h w'),
                    nn.Conv2d(1,1,2,4),
                    Rearrange('b 1 h w -> b h w'),
                    )
                self.hw_patch = int(self.hw_patch / 2)
            self.layers.append(transformer_block)
        self.head = nn.Sequential(nn.LayerNorm(emb_dim), nn.Linear(emb_dim, out_dim),nn.Softmax(dim=1))

    def forward(self, img):
        x = self.patch_embedding(img)
        b, n, _ = x.shape
        cls_tokens = repeat(self.cls_token, '1 1 d -> b 1 d', b = b)
        x = torch.cat([cls_tokens, x], dim=1)
        x += self.pos_embedding[:, :(n + 1)]
        for i in range(self.n_layers):
            if i == 0:
                x = self.layers[i](x)
                self.att_matrix[i] = x
            else:
                x = x[:,1:,:]
                x = self.layers[i][0](x)
                x = torch.cat([cls_tokens, x], dim=1)
                x = self.layers[i][1](x)
                att_in = x[:,1:,:]
                transformed_att = self.layers[i][3](att_in)
                if i > 1 :
                    self.att_matrix[i] = self.layers[i][4](self.att_matrix[i-1])
                    res_att= self.layers[i][5](self.att_matrix[i])
                    res_att = self.layers[i][6](res_att)
                    transformed_att += res_att
                self.att_matrix[i] = transformed_att
                b, n, e = transformed_att.shape
                transformed_att = torch.cat([torch.ones(b, 1, e).to(device), transformed_att], dim=1)
                b, n, e = transformed_att.shape
                transformed_att = torch.cat([torch.ones(b, n, 1).to(device), transformed_att], dim=2)
                x = self.layers[i][2](x)
                x = torch.matmul(nn.Softmax(dim=1)(transformed_att),x)
        self.attention_feature = x
        return self.head(x[:, 0, :])
```

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. TPAMI, pages 2481–2495, 2017. 1 [2] Han Cai, Chuang Gan, and Song Han. Efficientvit: Enhanced linear attention for high-resolution low-computation visual recognition. In IEEE/CVF ICCV, 2023. 6 [3] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. NeurIPS, pages 19974– 19988, 2021. 3 [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. In ICLR, 2021. 1, 2, 5, 6 [5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. TPAMI, pages 142–158, 2015. 1 [6] Qi Han, Zejia Fan, Qi Dai, Lei Sun, Ming-Ming Cheng, Jiaying Liu, and Jingdong Wang. Demystifying local vision transformer: Sparse connectivity, weight sharing, and dynamic weight. arXiv:2106.04263, 2021. 3 [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, pages 770–778, 2016. 4, 6 [8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Commun.ACM, pages 84–90, 2017. 1, 5 [9] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. arXiv:2202.07800, 2022. 1, 2, 6 [10] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and ´ C. Lawrence Zitnick. Microsoft COCO: common objects in context. In ECCV, pages 740–755, 2014. 5, 6 [11] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, ´ Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In CVPR, pages 2117–2125, 2017. 7 [12] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In ´ Proceedings of the IEEE international conference on computer vision, pages 2980–2988, 2017. 6, 7 [13] 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. In ICCV, pages 9992–10002, 2021. 6, 7 [14] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In CVPR, pages 12009–12019, 2022. 5 [15] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In CVPR, pages 11976–11986, 2022. 5 [16] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In ICLR, 2019. 6 [17] Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. Scalable vision transformers with hierarchical pooling. In ICCV, pages 377–386, 2021. 2, 6 [18] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. Fast vision transformers with hilo attention. NeurIPS, pages 14541– 14554, 2022. 2 [19] Zizheng Pan, Bohan Zhuang, Haoyu He, Jing Liu, and Jianfei Cai. Less is more: Pay less attention in vision transformers. In AAAI, pages 2035–2043, 2022. 2, 6 [20] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollar. Designing network design ´ spaces. In CVPR, pages 10428–10436, 2020. 6 [21] Yongming rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. NeurIPS, 34:13937–13949, 2021. 1, 2, 3, 6 [22] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In CVPR, pages 779–788, 2016. 1 [23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In MICCAI, pages 234–241, 2015. 1 [24] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In ICML, pages 6105–6114, 2019. 6 [25] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve J ´ egou. Training ´ data-efficient image transformers & distillation through attention. In ICML, pages 10347–10357, 2021. 2, 6 [26] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Herve J ´ egou. Going deeper with im- ´ age transformers. In ICCV, pages 32–42, 2021. 3, 4 [27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. NeurIPS, 30, 2017. 1, 2 [28] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In CVPR, pages 3156–3164, 2017. 1 [29] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions.

In ICCV, pages 568–578, 2021. 2, 6 [30] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvt v2: Improved baselines with pyramid vision transformer. Computational Visual Media, pages 415–424, 2022. 2, 6 [31] Cong Wei, Brendan Duke, Ruowei Jiang, Parham Aarabi, Graham W Taylor, and Florian Shkurti. Sparsifiner: Learning sparse instance-dependent attention for efficient vision transformers. In CVPR, pages 22680–22689, 2023. 3 [32] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In ICCV, pages 22–31, 2021. 2 [33] Xinjian Wu, Fanhu Zeng, Xiudong Wang, Yunhe Wang, and Xinghao Chen. Ppt: Token pruning and pooling for efficient vision transformers. arXiv:2310.01812, 2023. 1, 2, 6 [34] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In ECCV, pages 418–434, 2018. 7 [35] Jianwei Yang, Chunyuan Li, Pengchuan Zhang, Xiyang Dai, Bin Xiao, Lu Yuan, and Jianfeng Gao. Focal selfattention for local-global interactions in vision transformers. arXiv:2107.00641, 2021. 6 [36] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. In ICCV, pages 579–588, 2021. 2, 6 [37] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ADE20K dataset. Int. J. Comput. Vis., pages 302–321, 2019. 5 [38] Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. Deepvit: Towards deeper vision transformer. arXiv:2103.11886, 2021. 1, 3. [39] You Only Need Less Attention at Each Stage in Vision Transformers