AIL 862

Lecture 14

Context

- Global context

A bit about GNNs

Scaling

Self-Attention

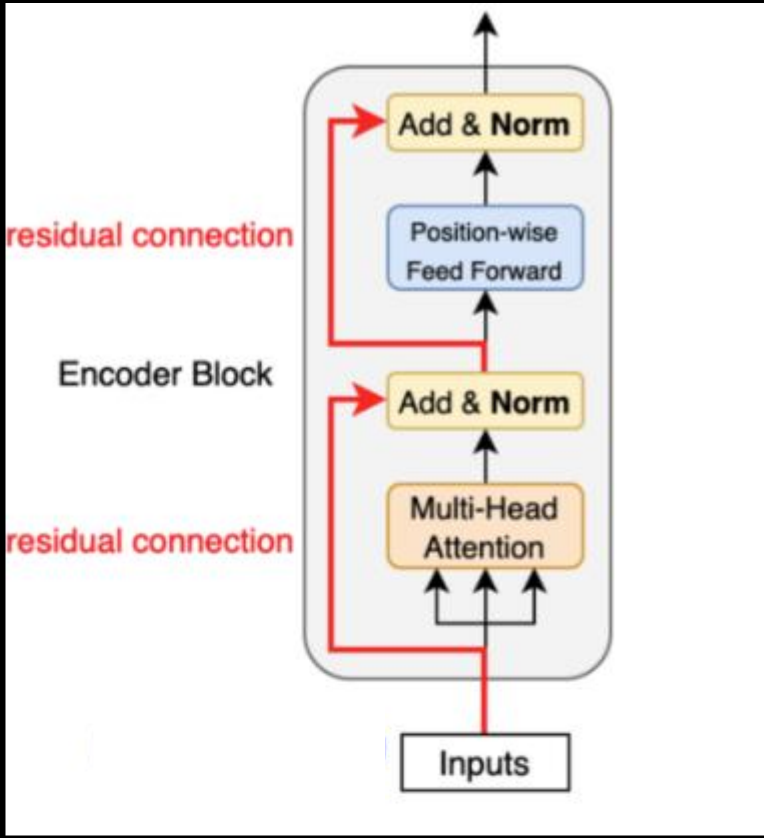
```python
class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64, dropout = 0.):
        super().__init__()
        inner_dim = dim_head *  heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.norm = nn.LayerNorm(dim)

        self.attend = nn.Softmax(dim = -1)
        self.dropout = nn.Dropout(dropout)

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        ) if project_out else nn.Identity()
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

```python
def forward(self, x):
    x = self.norm(x)

    qkv = self.to_qkv(x).chunk(3, dim = -1)
    q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h = self.heads), qkv)

    dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

    attn = self.attend(dots)
    attn = self.dropout(attn)

    out = torch.matmul(attn, v)
    out = rearrange(out, 'b h n d -> b n (h d)')
    return self.to_out(out)
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Transformer Encoder



```python
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout = 0.):
        super().__init__()
        self.norm = nn.LayerNorm(dim)
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                Attention(dim, heads = heads, dim_head = dim_head, dropout = dropout),
                FeedForward(dim, mlp_dim, dropout = dropout)
            ]))

    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x

        return self.norm(x)
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Image as a sequence

- Sequence of ?

Divide the input image

- Into patches

Embedding

- Treat the patches as embedding

Alternate embedding

- Extracted from a CNN

Sense of location

- Positional embedding

Process through

- Several transformer layers

Classification

- Take the final output and process it through fully connected layer

Which final output

One extra token

```python
class ViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads, mlp_dim, pool = 'cls', channels = 3, dim_head = 64, dropout = 0., emb_dropout = 0
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)

        assert image_height % patch_height == 0 and image_width % patch_width == 0, 'Image dimensions must be divisible by the patch size.'

        num_patches = (image_height // patch_height) * (image_width // patch_width)
        patch_dim = channels * patch_height * patch_width
        assert pool in {'cls', 'mean'}, 'pool type must be either cls (cls token) or mean (mean pooling)'

        self.to_patch_embedding = nn.Sequential(
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = patch_height, p2 = patch_width),
            nn.LayerNorm(patch_dim),
            nn.Linear(patch_dim, dim),
            nn.LayerNorm(dim),
        )

        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)

        self.pool = pool
        self.to_latent = nn.Identity()

        self.mlp_head = nn.Linear(dim, num_classes)
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

```python
def forward(self, img):
    x = self.to_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)]
    x = self.dropout(x)

    x = self.transformer(x)

    x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]

    x = self.to_latent(x)
    return self.mlp_head(x)
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

Different Variants

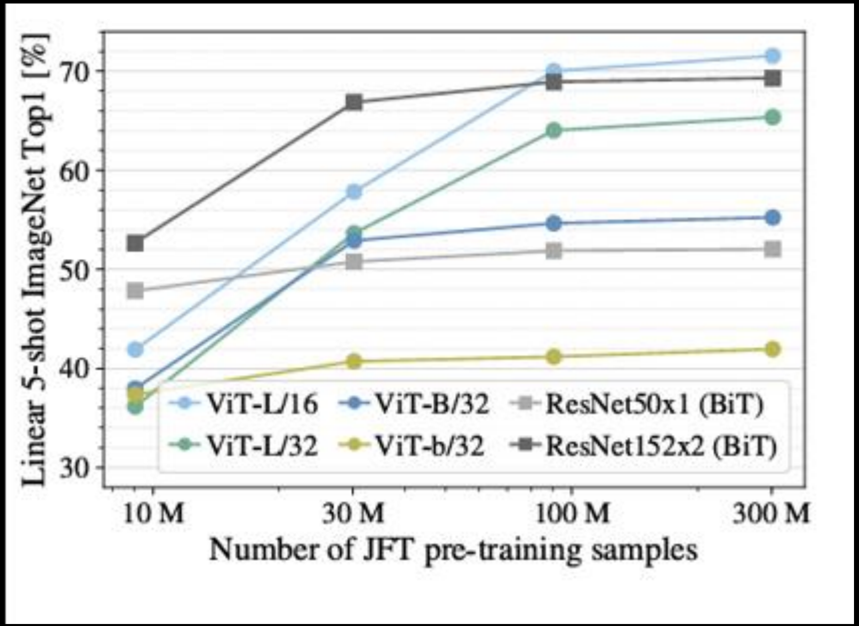
Model	Layers	Heads	Params
ViT-Base	12	12	86M
ViT-Large	24	16	307M
ViT-Huge	32	16	632M

Pre-Training Data Requirement

Pre-Training Data Requirement

- Pre-train on increasing size datasets (ImageNet, ImageNet-21K, JFT-300M)
- Fine tune on target dataset
- Observe peformance

Scaling



Linear 5-shot ImageNet Top1 [%]
70
60
50
40
30
10 M
30 M
100 M
300 M
Number of JFT pre-training samples
ViT-L/16
ViT-B/32
ResNet50x1 (BiT)
ViT-L/32
ViT-b/32
ResNet152x2 (BiT)