Introduction to Transformers

Mubarak Shah
shah@crcv.ucf.edu
Contents

• What is Transformer?
• Self-Attention
  • Query, Key, Value
• Position encoding
• Encoder-Decoder
Transformer

• Used for modeling long dependencies between input sequence elements
• Supports parallel processing of sequence as compared to RNN (e.g. LSTM)
• Requires minimal inductive biases for its design
• Allows processing multiple modalities
  • (e.g., images, videos, text and speech) using similar processing blocks
• Typically, pre-trained using pretext tasks on largescale (unlabeled) datasets
• Demonstrates excellent scalability to very large networks and huge datasets.
Vision Applications

- Recognition tasks (e.g., image classification, object detection, action recognition, and segmentation),
- Generative modeling, multi-modal tasks (e.g., visual-question answering, visual reasoning, and visual grounding),
- Video processing (e.g., activity recognition, video forecasting),
- Low-level vision (e.g., image super-resolution, image enhancement, and colorization)
- 3D analysis (e.g., point cloud classification and segmentation)
Natural Language Processing

• BERT (Bidirectional Encoder Representations from Transformers),
• GPT (Generative Pre-trained Transformer) v1-3,
• RoBERTa (Robustly Optimized BERT Pre-training)
• T5 (Text-to-Text Transfer Transformer)
BERT (Bidirectional Encoder Representations from Transformers)

• Original Transformer model could only attend to the context on the left of a given word
• BERT jointly encodes the right and left context of a word in a sentence to improve the learned feature representations
• BERT is trained on two pre-text tasks in self-supervised manner
  • Masked Language Model (MLM)
    • Mask fixed percentage (15%) of words in a sentence predict these masked words
    • In predicting the masked words, the model learns the bidirectional context.
  • Next Sentence Prediction (NSP)
    • Given a pair of sentences A and B the model predicts a binary label i.e., whether the pair is valid or not from the original document
    • Pair is formed such that B is the actual sentence (next to A) 50% of the time, and B is a random sentence for other 50% of the time.
Transformer

• It consists of Encoder and Decoder Blocks

• Main components of each block:
  • Self-Attention
  • Layer Normalization
  • Feed Forward Network
Slide courtesy of AI Bites, Youtube Channel: https://www.youtube.com/c/AIBites
Self-Attention

Slide courtesy of AI Bites, Youtube
Channel: https://www.youtube.com/c/AIBites
Self-Attention

\[ \begin{array}{cccc}
  x_1 & x_2 & x_3 & x_4 \\
  0.15 & 0.87 & 0.81 & 0.77 \\
  0.21 & 0.03 & 0.61 & 0.10 \\
  0.28 & 0.58 & 0.02 & 0.47 \\
  0.82 & 0.67 & 0.43 & 0.27 \\
\end{array} \]

Normalize

\[ \begin{array}{cccc}
  0.15 & 0.87 & 0.81 & 0.77 \\
  0.21 & 0.03 & 0.61 & 0.10 \\
  0.28 & 0.58 & 0.02 & 0.47 \\
  0.82 & 0.67 & 0.43 & 0.27 \\
\end{array} \]
### Self-Attention

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>0.67</td>
<td>0.81</td>
<td>0.77</td>
</tr>
<tr>
<td>0.21</td>
<td>0.03</td>
<td>0.91</td>
<td>0.10</td>
</tr>
<tr>
<td>0.28</td>
<td>0.58</td>
<td>0.02</td>
<td>0.47</td>
</tr>
<tr>
<td>0.82</td>
<td>0.67</td>
<td>0.43</td>
<td>0.27</td>
</tr>
</tbody>
</table>

\[ W \times X^T = Y \]
Self-Attention

\[ Y = W X^T \]

Slide courtesy of AI Bites, Youtube Channel: https://www.youtube.com/c/AIBites
Self-Attention

Let's denote a sequence of \( n \) entities \((x_1, x_2, \cdots x_n)\) by \( X \in \mathbb{R}^{n \times d} \).

Queries \( (W_Q^Q \in \mathbb{R}^{d \times d_q}) \), Keys \( (W_K^K \in \mathbb{R}^{d \times d_k}) \) and Values \( (W_V^V \in \mathbb{R}^{d \times d_v}) \), where \( d_q = d_k \). The input sequence \( X \) is first projected onto these weight matrices to get \( \hat{Q} = X W_Q^Q \), \( \hat{K} = X W_K^K \) and \( \hat{V} = X W_V^V \). The output \( Z = \text{softmax} \left( \frac{QK^T}{\sqrt{d_q}} \right) V \).
Self-Attention

Let's denote a sequence of $n$ entities $(x_1, x_2, \cdots x_n)$ by $X \in \mathbb{R}^{n \times d}$.

Queries ($W^Q \in \mathbb{R}^{d \times d_q}$), Keys ($W^K \in \mathbb{R}^{d \times d_k}$) and Values ($W^V \in \mathbb{R}^{d \times d_v}$), where $d_q = d_k$. The input sequence $X$ is first projected onto these weight matrices to get $Q = XW^Q$, $K = XW^K$ and $V = XW^V$. The output $Z \in \mathbb{R}^{n \times d_v}$ of the

$$Z = \text{softmax} \left( \frac{QK^T}{\sqrt{d_q}} \right) V.$$
Transformers (Attention is all you need 2017)


- Two valuable sources
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - https://jalammar.github.io/illustrated-transformer/ (slides come from this source)

- Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Attention and Transformers

Transformer

INPUT: Je suis étudiant
OUTPUT: I am a student

Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Attention and Transformers

An Encoder Block: same structure, different parameters

Feed Forward Neural Network

Self-Attention

Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Attention and Transformers

The ffnn is independent for each word. Hence can be parallelized.
Self Attention

- First, we create three vectors by multiplying input embedding $x_i$ with three matrices
  - $q_i = x_i W^Q$
  - $K_i = x_i W^K$
  - $V_i = x_i W^V$
Self Attention

Now we calculate a score to determine how much focus to place on other Parts of the input.

<table>
<thead>
<tr>
<th>Input</th>
<th>Thinking</th>
<th>Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding</td>
<td>$x_1$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Queries</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>Keys</td>
<td>$k_1$</td>
<td>$k_2$</td>
</tr>
<tr>
<td>Values</td>
<td>$v_1$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>Score</td>
<td>$q_1 \cdot k_1 = 112$</td>
<td>$q_1 \cdot k_2 = 96$</td>
</tr>
</tbody>
</table>
Attention and Transformer

Self Attention

\[ \begin{align*}
\text{Input} & \\
\text{Embedding} & \\
\text{Queries} & \\
\text{Keys} & \\
\text{Values} & \\
\text{Score} & \\
\text{Divide by } 8 (\sqrt{d_k}) & \\
\text{Softmax} & \\
\text{Softmax} \times \text{X} & \\
\text{Value} & \\
\text{Sum} & \\
\end{align*} \]

\[ z_1 = 0.88v_1 + 0.12v_2 \]

Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Multiple heads

1. It expands the model’s ability to focus on different positions.
2. It gives the attention layer multiple “representation subspaces”
02

Attention and Transformers

1) Concatenate all the attention heads

\[
\begin{array}{cccccccc}
Z_0 & Z_1 & Z_2 & Z_3 & Z_4 & Z_5 & Z_6 & Z_7 \\
\end{array}
\]

2) Multiply with a weight matrix \( W^o \) that was trained jointly with the model

\[
\begin{array}{cccccccc}

\end{array}
\]

3) The result would be the \( z \) matrix that captures information from all the attention heads. We can send this forward to the FFNN

\[
\begin{array}{cccccccc}
Z
\end{array}
\]
1) This is our input sentence
2) We embed each word
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^o to produce the output of the layer
Layer: 5  
Attention: Input - Input

The_
animal_
didn_
't_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Attention and Transformers

Representing the input order (positional encoding)

- Transformer is permutation invariant
- The transformer adds a vector to each input embedding.
- These vectors follow a specific pattern that the model learns.
- Learned pattern helps model
  - to determine the position of each word, or
  - the distance between different words.
Attention and Transformers

Representing the input order (positional encoding)
Position Encoding

\[ \vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} 
 \sin(\omega_k t), & \text{if } i = 2k \\
 \cos(\omega_k t), & \text{if } i = 2k + 1 
\end{cases} \]

where

\[ \omega_k = \frac{1}{10000^{2k/d}} \]

\[ \vec{p}_t = \begin{bmatrix} 
 \sin(\omega_1 t) \\
 \cos(\omega_1 t) \\
 \sin(\omega_2 t) \\
 \cos(\omega_2 t) \\
 \vdots \\
 \sin(\omega_{d/2} t) \\
 \cos(\omega_{d/2} t) 
\end{bmatrix}_{d \times 1} \]
Position Encoding
\[
P(k, 2i) = \sin \left( \frac{k}{n^{2i/d}} \right) \\
P(k, 2i + 1) = \cos \left( \frac{k}{n^{2i/d}} \right)
\]

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Index of token, ( k )</th>
<th>( i = 0 )</th>
<th>( i = 0 )</th>
<th>( i = 1 )</th>
<th>( i = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>( P_{00} = \sin(0) ) = 0</td>
<td>( P_{01} = \cos(0) ) = 1</td>
<td>( P_{02} = \sin(0) ) = 0</td>
<td>( P_{03} = \cos(0) ) = 1</td>
</tr>
<tr>
<td>am</td>
<td>1</td>
<td>( P_{10} = \sin(1/1) ) = 0.84</td>
<td>( P_{11} = \cos(1/1) ) = 0.54</td>
<td>( P_{12} = \sin(1/10) ) = 0.10</td>
<td>( P_{13} = \cos(1/10) ) = 1.0</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>( P_{20} = \sin(2/1) ) = 0.91</td>
<td>( P_{21} = \cos(2/1) ) = -0.42</td>
<td>( P_{22} = \sin(2/10) ) = 0.20</td>
<td>( P_{23} = \cos(2/10) ) = 0.98</td>
</tr>
<tr>
<td>Robot</td>
<td>3</td>
<td>( P_{30} = \sin(3/1) ) = 0.14</td>
<td>( P_{31} = \cos(3/1) ) = -0.99</td>
<td>( P_{32} = \sin(3/10) ) = 0.30</td>
<td>( P_{33} = \cos(3/10) ) = 0.96</td>
</tr>
</tbody>
</table>

Positional Encoding Matrix for the sequence 'I am a robot'
Position Encoding

- Can also be learned
- Learn like other parameters
Attention and Transformers

Add and Normalize

Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Layer Normalization (Hinton)

Layer normalization normalizes the inputs across the features.
The complete transformer

Attention and Transformers

Slides from Ming Li, University of Waterloo, CS 886 Deep Learning and NLP
Vision Transformers

Mubarak Shah
shah@crcv.ucf.edu
Vision Transformer (VIT)

Vision Transformer (ViT)

• Naive application of self-attention to images requires high computation
• Divide an image into 16x16 patches (tokens)
• Transformers need to be trained on large datasets
• ViT attains excellent results when pre-trained on JFT-300M
  • 88:55% on ImageNet,
  • 90:72% on ImageNet-ReaL,
  • 94:55% on CIFAR-100,
  • 77:63% on the VTAB suite of 19 tasks
Vision Transformer (ViT)
Divide image into 16x16 patches
Generate embedding for each patch
CLS (Classification) Token
Complete VIT

Slide credit: Piotr Mazurek
VIT Model Variants

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Hidden size $D$</th>
<th>MLP size</th>
<th>Heads</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-Base</td>
<td>12</td>
<td>768</td>
<td>3072</td>
<td>12</td>
<td>86M</td>
</tr>
<tr>
<td>ViT-Large</td>
<td>24</td>
<td>1024</td>
<td>4096</td>
<td>16</td>
<td>307M</td>
</tr>
<tr>
<td>ViT-Huge</td>
<td>32</td>
<td>1280</td>
<td>5120</td>
<td>16</td>
<td>632M</td>
</tr>
</tbody>
</table>

Table 1: Details of Vision Transformer model variants.
## Results

<table>
<thead>
<tr>
<th></th>
<th>Ours-JFT (ViT-H/14)</th>
<th>Ours-JFT (ViT-L/16)</th>
<th>Ours-121k (ViT-L/16)</th>
<th>BiT-L (ResNet152x4)</th>
<th>Noisy Student (EfficientNet-L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet</td>
<td>88.55 ± 0.04</td>
<td>87.76 ± 0.03</td>
<td>85.30 ± 0.02</td>
<td>87.54 ± 0.02</td>
<td>88.4/88.5*</td>
</tr>
<tr>
<td>ImageNet ReAL</td>
<td>90.72 ± 0.05</td>
<td>90.54 ± 0.03</td>
<td>88.62 ± 0.05</td>
<td>90.54</td>
<td>90.55</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>99.50 ± 0.06</td>
<td>99.42 ± 0.03</td>
<td>99.15 ± 0.03</td>
<td>99.37 ± 0.06</td>
<td>—</td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>94.55 ± 0.04</td>
<td>93.90 ± 0.05</td>
<td>93.25 ± 0.05</td>
<td>93.51 ± 0.08</td>
<td>—</td>
</tr>
<tr>
<td>Oxford-IIIT Pets</td>
<td>97.56 ± 0.03</td>
<td>97.32 ± 0.11</td>
<td>94.67 ± 0.15</td>
<td>96.62 ± 0.23</td>
<td>—</td>
</tr>
<tr>
<td>Oxford Flowers-102</td>
<td>99.68 ± 0.02</td>
<td>99.74 ± 0.00</td>
<td>99.61 ± 0.02</td>
<td>99.63 ± 0.03</td>
<td>—</td>
</tr>
<tr>
<td>VTAB (19 tasks)</td>
<td>77.63 ± 0.23</td>
<td>76.28 ± 0.46</td>
<td>72.72 ± 0.21</td>
<td>76.29 ± 1.70</td>
<td>—</td>
</tr>
<tr>
<td>TPUv3-core-days</td>
<td>2.5k</td>
<td>0.68k</td>
<td>0.23k</td>
<td>9.9k</td>
<td>12.3k</td>
</tr>
</tbody>
</table>
Attention
Vision Transformer (ViT)
Cross Entropy and KL (Kullback-Leibler) divergence

- **Entropy**: \( E(P) = - \sum_i P(i) \log P(i) \)
- **Cross Entropy**: \( C(P) = - \sum_i P(i) \log Q(i) \)
- **KL divergence**: \( D_{KL}(P \| Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} = \sum_i P(i) [\log P(i) - \log Q(i)] \)
- **JSD** (P||Q) = \( \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M) \), \( M = \frac{1}{2} (P+Q) \), symmetric KL

* JSD = Jensen-Shannon Divergency
SWIN

• Z Liu, Y Lin, Y Cao, H Hu, Y Wei, Z Zhang, S Lin, “Swin transformer: Hierarchical vision transformer using shifted windows”, ICCV-2021. (Marr Prize) 4693 Citations
SWIN

• Adapting Transformer from language to vision is challenging

• Differences between language and vision Domains
  • Large variations in the scale of visual entities
  • High resolution of pixels in images compared to words in text

• To address these differences, SWIN proposes
  • A hierarchical Transformer whose representation is computed with Shifted windows
  • Shifted Windowing limit attention
    • To local windows and
    • Allowing cross window connections
SWIN

• 87.3 top-1 accuracy on ImageNet-1K
• Dense prediction tasks such as
  • Object detection (58.7 box AP and 51.1 mask AP on COCO)
  • Semantic segmentation (53.5 mIoU on ADE20K)
• Performance surpasses the previous state-of-the art by
  • +2.7 box AP and +2.6 mask AP on COCO, and
  • +3.2 mIoU on ADE20K.
Hierarchical Feature Maps and Local Attention

(a) Swin Transformer (ours)

(b) ViT
SWIN Architecture

C=96, 128, 192

(a) Architecture

Two Successive Swin Transformer Blocks
Self-Attention within each window and shifted windows
Transformer Blocks

\[
\hat{z}^l = W\text{-MSA} \left( \text{LN} \left( z^{l-1} \right) \right) + z^{l-1},
\]
\[
z^l = \text{MLP} \left( \text{LN} \left( \hat{z}^l \right) \right) + \hat{z}^l,
\]
\[
\hat{z}^{l+1} = \text{SW-MSA} \left( \text{LN} \left( z^l \right) \right) + z^l,
\]
\[
z^{l+1} = \text{MLP} \left( \text{LN} \left( \hat{z}^{l+1} \right) \right) + \hat{z}^{l+1},
\]
Different Configurations

- Swin-T: $C = 96$, layer numbers = $\{2, 2, 6, 2\}$
- Swin-S: $C = 96$, layer numbers = $\{2, 2, 18, 2\}$
- Swin-B: $C = 128$, layer numbers = $\{2, 2, 18, 2\}$
- Swin-L: $C = 192$, layer numbers = $\{2, 2, 18, 2\}$
## Results

### (a) Regular ImageNet-1K trained models

<table>
<thead>
<tr>
<th>method</th>
<th>image size</th>
<th>#param.</th>
<th>FLOPs</th>
<th>throughput (image / s)</th>
<th>ImageNet top-1 acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegNetY-4G [44]</td>
<td>224²</td>
<td>21M</td>
<td>4.0G</td>
<td>1156.7</td>
<td>80.0</td>
</tr>
<tr>
<td>RegNetY-8G [44]</td>
<td>224²</td>
<td>39M</td>
<td>8.0G</td>
<td>591.6</td>
<td>81.7</td>
</tr>
<tr>
<td>RegNetY-16G [44]</td>
<td>224²</td>
<td>84M</td>
<td>16.0G</td>
<td>334.7</td>
<td>82.9</td>
</tr>
<tr>
<td>ViT-B/16 [19]</td>
<td>384²</td>
<td>86M</td>
<td>55.4G</td>
<td>85.9</td>
<td>77.9</td>
</tr>
<tr>
<td>ViT-L/16 [19]</td>
<td>384²</td>
<td>307M</td>
<td>190.7G</td>
<td>27.3</td>
<td>76.5</td>
</tr>
<tr>
<td>DeiT-S [57]</td>
<td>224²</td>
<td>22M</td>
<td>4.6G</td>
<td>940.4</td>
<td>79.8</td>
</tr>
<tr>
<td>DeiT-B [57]</td>
<td>224²</td>
<td>86M</td>
<td>17.5G</td>
<td>292.3</td>
<td>81.8</td>
</tr>
<tr>
<td>DeiT-B [57]</td>
<td>384²</td>
<td>86M</td>
<td>55.4G</td>
<td>85.9</td>
<td>83.1</td>
</tr>
<tr>
<td>Swin-T</td>
<td>224²</td>
<td>29M</td>
<td>4.5G</td>
<td>755.2</td>
<td>81.3</td>
</tr>
<tr>
<td>Swin-S</td>
<td>224²</td>
<td>50M</td>
<td>8.7G</td>
<td>436.9</td>
<td>83.0</td>
</tr>
<tr>
<td>Swin-B</td>
<td>224²</td>
<td>88M</td>
<td>15.4G</td>
<td>278.1</td>
<td>83.5</td>
</tr>
<tr>
<td>Swin-B</td>
<td>384²</td>
<td>88M</td>
<td>47.0G</td>
<td>84.7</td>
<td>84.5</td>
</tr>
</tbody>
</table>
## Results

### (b) ImageNet-22K pre-trained models

<table>
<thead>
<tr>
<th>method</th>
<th>image size</th>
<th>#param.</th>
<th>FLOPs</th>
<th>throughput (image / s)</th>
<th>ImageNet top-1 acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-101x3 [34]</td>
<td>384²</td>
<td>388M</td>
<td>204.6G</td>
<td>-</td>
<td>84.4</td>
</tr>
<tr>
<td>R-152x4 [34]</td>
<td>480²</td>
<td>937M</td>
<td>840.5G</td>
<td>-</td>
<td>85.4</td>
</tr>
<tr>
<td>ViT-B/16 [19]</td>
<td>384²</td>
<td>86M</td>
<td>55.4G</td>
<td>85.9</td>
<td>84.0</td>
</tr>
<tr>
<td>ViT-L/16 [19]</td>
<td>384²</td>
<td>307M</td>
<td>190.7G</td>
<td>27.3</td>
<td>85.2</td>
</tr>
<tr>
<td>Swin-B</td>
<td>224²</td>
<td>88M</td>
<td>15.4G</td>
<td>278.1</td>
<td>85.2</td>
</tr>
<tr>
<td>Swin-B</td>
<td>384²</td>
<td>88M</td>
<td>47.0G</td>
<td>84.7</td>
<td>86.4</td>
</tr>
<tr>
<td>Swin-L</td>
<td>384²</td>
<td>197M</td>
<td>103.9G</td>
<td>42.1</td>
<td>87.3</td>
</tr>
</tbody>
</table>
## Results

### Table 3: Results of semantic segmentation on the ADE20K val set

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>val mIoU</th>
<th>test score</th>
<th>param</th>
<th>FLOPs</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNL [65]</td>
<td>ResNet-101</td>
<td>46.0</td>
<td>56.2</td>
<td>69M</td>
<td>1249G</td>
<td>14.8</td>
</tr>
<tr>
<td>OCRNet [67]</td>
<td>ResNet-101</td>
<td>45.3</td>
<td>56.0</td>
<td>56M</td>
<td>923G</td>
<td>19.3</td>
</tr>
<tr>
<td>UperNet [63]</td>
<td>ResNet-101</td>
<td>44.9</td>
<td>-</td>
<td>86M</td>
<td>1029G</td>
<td>20.1</td>
</tr>
<tr>
<td>OCRNet [67]</td>
<td>HRNet-w48</td>
<td>45.7</td>
<td>-</td>
<td>71M</td>
<td>664G</td>
<td>12.5</td>
</tr>
<tr>
<td>DLab.v3+ [11]</td>
<td>ResNeSt-101</td>
<td>46.9</td>
<td>55.1</td>
<td>66M</td>
<td>1051G</td>
<td>11.9</td>
</tr>
<tr>
<td>DLab.v3+ [11]</td>
<td>ResNeSt-200</td>
<td>48.4</td>
<td>-</td>
<td>88M</td>
<td>1381G</td>
<td>8.1</td>
</tr>
<tr>
<td>SETR [73]</td>
<td>T-Large</td>
<td>50.3</td>
<td>61.7</td>
<td>308M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>UperNet</td>
<td>DeiT-S</td>
<td>44.0</td>
<td>-</td>
<td>52M</td>
<td>1099G</td>
<td>16.2</td>
</tr>
<tr>
<td>UperNet</td>
<td>Swin-T</td>
<td>46.1</td>
<td>-</td>
<td>60M</td>
<td>945G</td>
<td>18.5</td>
</tr>
<tr>
<td>UperNet</td>
<td>Swin-S</td>
<td>49.3</td>
<td>-</td>
<td>81M</td>
<td>1038G</td>
<td>15.2</td>
</tr>
<tr>
<td>UperNet</td>
<td>Swin-B</td>
<td>51.6</td>
<td>-</td>
<td>121M</td>
<td>1841G</td>
<td>8.7</td>
</tr>
<tr>
<td>UperNet</td>
<td>Swin-L</td>
<td>53.5</td>
<td>62.8</td>
<td>234M</td>
<td>3230G</td>
<td>6.2</td>
</tr>
</tbody>
</table>

### Table 4: Various backbones w. Cascade Mask R-CNN

<table>
<thead>
<tr>
<th>Method</th>
<th>AP_{box}</th>
<th>AP_{50}</th>
<th>AP_{75}</th>
<th>AP_{mask}</th>
<th>AP_{50}</th>
<th>AP_{75}</th>
<th>param</th>
<th>FLOPs</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeiT-S</td>
<td>48.0</td>
<td>67.2</td>
<td>51.7</td>
<td>41.4</td>
<td>64.2</td>
<td>44.3</td>
<td>80M</td>
<td>889G</td>
<td>10.4</td>
</tr>
<tr>
<td>R50</td>
<td>46.3</td>
<td>64.3</td>
<td>50.4</td>
<td>41.0</td>
<td>61.1</td>
<td>43.4</td>
<td>82M</td>
<td>739G</td>
<td>18.0</td>
</tr>
<tr>
<td>Swin-T</td>
<td><strong>50.5</strong></td>
<td><strong>69.3</strong></td>
<td><strong>54.9</strong></td>
<td><strong>43.7</strong></td>
<td><strong>66.6</strong></td>
<td><strong>47.1</strong></td>
<td>86M</td>
<td>745G</td>
<td>15.3</td>
</tr>
<tr>
<td>X101-32</td>
<td>48.1</td>
<td>66.5</td>
<td>52.4</td>
<td>41.3</td>
<td>62.9</td>
<td>44.6</td>
<td>101M</td>
<td>819G</td>
<td>12.8</td>
</tr>
<tr>
<td>Swin-S</td>
<td><strong>51.8</strong></td>
<td><strong>70.4</strong></td>
<td><strong>56.3</strong></td>
<td><strong>44.7</strong></td>
<td><strong>67.9</strong></td>
<td><strong>48.5</strong></td>
<td>107M</td>
<td>838G</td>
<td>12.0</td>
</tr>
<tr>
<td>X101-64</td>
<td>48.3</td>
<td>66.9</td>
<td>52.3</td>
<td>41.7</td>
<td>64.0</td>
<td>45.1</td>
<td>140M</td>
<td>972G</td>
<td>10.4</td>
</tr>
<tr>
<td>Swin-B</td>
<td><strong>51.9</strong></td>
<td><strong>70.9</strong></td>
<td><strong>56.5</strong></td>
<td><strong>45.0</strong></td>
<td><strong>68.4</strong></td>
<td><strong>48.7</strong></td>
<td>145M</td>
<td>982G</td>
<td>11.6</td>
</tr>
</tbody>
</table>
Summary

• VIT is first Vision Transformer, but trained on huge dataset of 300M

• SWIN employs window attention, in addition performs well on other tasks: object detection, semantic segmentation