An Image is 16×16 words: Transformers for Image Recognition at Scale

- [[📚 2010.11929.pdf]]
- **Basic Idea**:
  - Can we use Transformers for Vision task without much changes?
    - 🟡 **P4** When considering the computational cost of pre-training the model, ViT performs very favourably, attaining state of the art on most recognition benchmarks at a lower pre-training cost.
  - Challenges with Transformers wrt CNNs:
    - 🟡 **P1** Transformers lack some of the inductive biases 🟡 **P2** inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data.
      - *One way to overcome this*: 🟡 **P2** Parmar et al. (2018) applied the self-attention only in local neighborhoods for each query pixel instead of globally. Such local multi-head dot-product self attention blocks can completely replace convolutions
    - Words in NLP have semantic meanings, whereas pixels are just raw numbers. Hence, how to perform self-attention on pixels because quadratic cost of self-attention would be huge for pixels. Idea: Convert images into patches and consider each patch as a word.

- **Vision Transformer (ViT)**
  - 🟡 **P3**

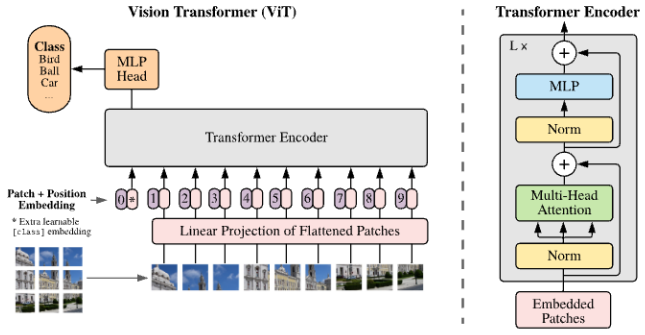


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

  - **Generating patches**:
    - Input image $x \in \mathbb{R}^{H \times W \times C} \implies$ Flatten to $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$ where
      - (H, W) is the input image resolution,
      - C is # channels,
      - (P, P) is the resolution of each image patch and
      - $N = HW/P^2$ is the # patches = Input sequence length to the Transformer.
    - 🟡 **P5** Note that the Transformer's sequence length is inversely proportional to the square of the patch size, thus models with smaller patch size are computationally more expensive
    - *Patch Embeddings* $\mathbf{E}$: Linear projection of the patches into fixed dimensional latent vector of D dimension to be used throughout the Network. $\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$.
    - 🟡 **P3** Similar to BERT's [class] token, we prepend a learnable embedding to the sequence of embedded patches (z00 = xclass), whose state at the output of the Transformer encoder (z0 L) serves as the image representation y
    - *Position Embeddings* $\mathbf{E}_{pos}$: $\mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$
    - 🟡 **P4**

The MLP contains two layers with a GELU non-linearity.

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \ldots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

    - **Food for thought**:
      - Does the aspect ratio of the patches matter?

- Converting images into patches is similar to converting images into grid for YOLO. Therefore, using ViTs for Object Detection should be straight-forward.
- Patch Embeddings are similar to convolutions. Since, the weights are shared across patches, there is locality and translation equivariance.

- **Inductive Bias**:
  - CNNs: Locality, 2D neighbourhood structure, translation equivariance
  - ViT: Only MLP layers are local and translation equivariant (Remember Position-wise feed forward neural networks in Transformers). This is because the weight matrix of this feed forward network is shared between the positions i.e., the MLP layers in Transformers are similar to convolution.
  - 🟡 **P4** The two-dimensional neighborhood structure is used very sparingly: in the beginning of the model by cutting the image into patches and at fine-tuning time for adjusting the position embeddings for images of different resolution (as described below). Other than that, the position embeddings at initialization time carry no information about the 2D positions of the patches and all spatial relations between the patches have to be learned from scratch

- **Hybrid Architecture**:
  - 🟡 **P4** Hybrid Architecture. As an alternative to raw image patches, the input sequence can be formed from feature maps of a CNN (LeCun et al., 1989). In this hybrid model, the patch embedding projection E (Eq. 1) is applied to patches extracted from a CNN feature map. As a special case, the patches can have spatial size 1×1, which means that the input sequence is obtained by simplyflattening the spatial dimensions of the feature map and projecting to the Transformer dimension. The classification input embedding and position embeddings are added as described above
  - **Food for thought**: Connection of this hybrid architecture to the Conditional Random Fields (CRFs) based optimization used in early segmentation networks to improve the segmentation masks (DeepLab v1)?

- **Fine-tuning**
  - While pre-training they use a hidden-layer followed by a classification layer (2 fc layers) on top of z_L^0 (the image representation y), whereas for fine-tuning they use a single fc layer which directly classifies into the target K classes. 🟡 **P4** we remove the pre-trained prediction head and attach a zero-initialized D × K feedforward layer, where K is the number of downstream classes
  - Based on previous studies, they fine-tune at a higher resolution. When training at higher resolution, they keep the patch size to be the same, thereby increasing the sequence length. 🟡 **P4** The Vision Transformer can handle arbitrary sequence lengths (up to memory constraints), however, the pre-trained position embeddings may no longer be meaningful. We therefore perform2D interpolation of the pre-trained position embeddings, according to their location in the original image. Note that this resolution adjustment and patch extraction are the only points at which an inductive bias about the 2D structure of the images is manually injected into the Vision Transformer

- **Results**
  - **Notations**: 🟡 **P5** In what follows we use brief notation to indicate the model size and the input patch size: for instance, ViT-L/16 means the "Large" variant with 16 × 16 input patch size.

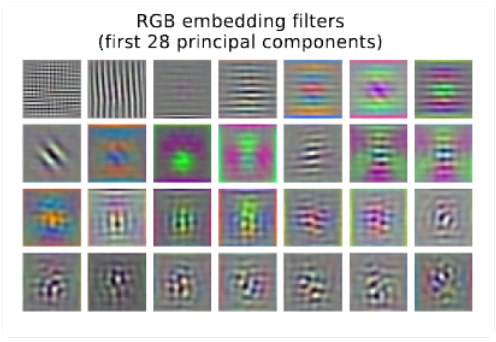
  - **Tricks of the Trade**:
    - *Batch Normalization vs Group Normalization*: 🟡 **P5** For the baseline CNNs, we use ResNet (He et al., 2016), but replace the Batch Normalization layers (Ioffe & Szegedy, 2015) with Group Normalization (Wu & He, 2018),
    - *Standardized Convolutions*: 🟡 **P5** used standardized convolutions (Qiao et al., 2019). These modifications improve transfer (Kolesnikov et al., 2020), and we denote the modified model "ResNet (BiT)".
    - Adam instead of SGD for fine-tuning ResNets
    - *Optimization*: 🟡 **P5** Polyak & Juditsky (1992) averaging with a factor of 0.9999 (Ramachandran et al., 2019; Wang et al., 2020b).
    - 🟡 **P13** Dropout, when used, is applied after every dense layer except for the the qkv-projections and directly after adding positional- to patch embeddings.
    - 🟡 **P13** For ImageNet we found it beneficial to additionally apply gradient clipping at global norm 1.

  - *ViTs have fewer inductive biases than ResNets*: 🟡 **P7** he convolutional inductive bias is useful for smaller datasets, but for larger ones, learning the relevant patterns directly from data is sufficient,

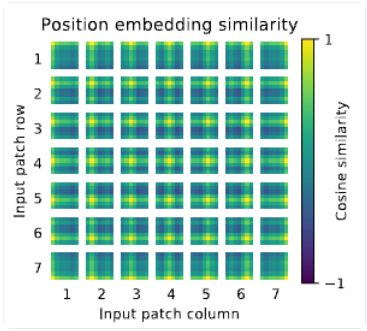
even beneficial.

- *Scaling Study*: Keeping the dataset size fixed, assess performance versus pre-training cost of each model:
  - 🟡 **P8** First, Vision Transformers dominate ResNets on the performance/compute trade-off. ViT uses approximately 2 – 4× less compute to attain the same performance (average over 5 datasets
  - 🟡 **P8** Second, hybrids slightly outperform ViT at small computational budgets, but the difference vanishes for larger models. This result is somewhat surprising, since one might expect convolutional local feature processing to assist ViT at any size
  - 🟡 **P8** Third, Vision Transformers appear not to saturate within the range tried, motivating future scaling efforts
  - Scaling depth is better than scaling width of the network.
  - 🟡 **P16** Decreasing the patch size and thus increasing the effective sequence length shows surprisingly robust improvements without introducing parameters.

- *Visual Inspection*:
  - *Patch Embeddings*:
    - 🟡 **P9**



    - Patch embeddings can be thought of as convolutions. $\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$, therefore, this layer can be thought of as learning D filters, each of shape $P \times P \times C$. Now, usually D is as high as 768, therefore, to visualize the important filters, we can do a PCA and pick first 28 filters.
    - 🟡 **P8** The components resemble plausible basis functions for a low-dimensional representation of the fine structure within each patch.
  - *Position Embeddings*:
    - 🟡 **P9**



    - 🟡 **P9** Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches.
    - 🟡 **P8** model learns to encode distance within the image in the similarity of position embeddings, i.e. closer patches tend to have more similar position embeddings.
    - 🟡 **P8** Further, the row-column structure appears; patches in the same row/column have similar embeddings.
    - 🟡 **P8** Finally, a sinusoidal structure is sometimes apparent for larger grids (Appendix D). That the position embeddings learn to represent 2D image topology explains why hand-crafted 2D-aware embedding variants do not yield improvements (Appendix D.4).
  - *Self-Attention*:

- 🟡 **P13**

  Standard **qkv** self-attention (SA, Vaswani et al. (2017)) is a popular building block for neural architectures. For each element in an input sequence $\mathbf{z} \in \mathbb{R}^{N \times D}$, we compute a weighted sum over all values $\mathbf{v}$ in the sequence. The attention weights A_{ij} are based on the pairwise similarity between two elements of the sequence and their respective query $\mathbf{q}^i$ and key $\mathbf{k}^j$ representations.

  $$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U}_{qkv} \qquad \mathbf{U}_{qkv} \in \mathbb{R}^{D \times 3D_h}, \tag{5}$$

  $$A = \operatorname{softmax}\left(\mathbf{q}\mathbf{k}^\top / \sqrt{D_h}\right) \qquad A \in \mathbb{R}^{N \times N}, \tag{6}$$

  $$\mathrm{SA}(\mathbf{z}) = A\mathbf{v}\,. \tag{7}$$

  Multihead self-attention (MSA) is an extension of SA in which we run k self-attention operations, called "heads", in parallel, and project their concatenated outputs. To keep compute and number of parameters constant when changing k, D_h (Eq. 5) is typically set to D/k.

  $$\mathrm{MSA}(\mathbf{z}) = [\mathrm{SA}_1(z); \mathrm{SA}_2(z); \cdots; \mathrm{SA}_k(z)]\,\mathbf{U}_{msa} \qquad \mathbf{U}_{msa} \in \mathbb{R}^{k \cdot D_h \times D} \tag{8}$$

  - *Attention Distance*: 🟡 **P8** the average distance in image space across which information is integrated, based on the attention weights. This is similar to receptive field size in CNNs.
  - 🟡 **P8** Self-attention allows ViT to integrate information across the entire image even in the lowest layers. 🟡 **P8** We find that some heads attend to most of the image already in the lowest layers, showing that the ability to integrate information globally is indeed used by the model.
  - 🟡 **P8** This highly localized attention is less pronounced in hybrid models that apply a ResNet before the Transformer (Figure 7, right), suggesting that it may serve a similar function as early convolutional layers in CNNs.
  - 🟡 **P8** Further, the attention distance increases with network depth
  - 🟡 **P20** Average attention distance is highly variable across heads in lower layers, with some heads attending to much of the image, while others attend to small regions at or near the query location. As depth increases, attention distance increases for all heads. In the second half of the network, most heads attend widely across tokens.
  - **Food for thought**: Does this indicate about hierarchical aggregation of information in ViTs similar to that in CNNs?

- *Self-supervision*:
  - Setup is similar to BERT (*Masked patch prediction*): 🟡 **P14** corrupt 50% of patch embeddings by either replacing their embeddings with a learnable[mask] embedding (80%), a random other patch embedding (10%) or just keeping them as is(10%).
  - *Prediction*: 🟡 **P14** Finally, we predict the 3-bit, mean color (i.e., 512 colors in total) of every corrupted patch using their respective patch representations.
    - For each 16×16 patch, they predict the mean colour which is represented by 3 bits. So 3 colours (RGB) of 3 bits i.e., $2^3 \times 2^3 \times 2^3 = 512$ possibilities.
    - Notice that the colours is reduced from 8-bit (0-255) to 3-bits (0-7). This can be done using `posterize augmentation` in albumentations. You can also do it using k-means clustering to determine the 8 cluster centres of the colours and assigning each pixel to one of the cluster centres.
    - Notice that instead of formulating the task as a regression task they have formulated it as a classification task and to keep the number of classes tractable they have reduced the number of colour bits. If the number of classes were high, some tricks like hierarchical softmax or contrastive loss from the word2vec could be used.
    - Other ways to define the prediction target (all of the methods give similar performance except L2 which is slightly worse):
      - 🟡 **P14** predicting a 4 × 4 downsized version of the 16 × 16 patch with 3bit colors in parallel (i.e., 16 predictions of 512 colors),
      - 🟡 **P14** regression on the full patch using L2 (i.e., 256 regressions on the 3 RGB channels)

- *Positional Embeddings*:
  - Ways of encoding positional/spatial information:

🟡 P17

- Providing no positional information: Considering the inputs as a *bag of patches*.
- 1-dimensional positional embedding: Considering the inputs as a sequence of patches in the raster order (default across all other experiments in this paper).
- 2-dimensional positional embedding: Considering the inputs as a grid of patches in two dimensions. In this case, two sets of embeddings are learned, each for one of the axes, X-embedding, and Y-embedding, each with size $D/2$. Then, based on the coordinate on the path in the input, we concatenate the X and Y embedding to get the final positional embedding for that patch.
- Relative positional embeddings: Considering the relative distance between patches to encode the spatial information as instead of their absolute position. To do so, we use 1-dimensional Relative Attention, in which we define the relative distance all possible pairs of patches. Thus, for every given pair (one as query, and the other as key/value in the attention mechanism), we have an offset $p_q - p_k$, where each offset is associated with an embedding. Then, we simply run extra attention, where we use the original query (the content of query), but use relative positional embeddings as keys. We then use the logits from the relative attention as a bias term and add it to the logits of the main attention (content-based attention) before applying the softmax.

- Ways of incorporating this information into the network:
  - 🟡 P17 (1) add positional embeddings to the inputs right after 🟡 P18 the stem of them model and before feeding the inputs to the Transformer encoder (default across all other experiments in this paper); (Adding positional embeddings to the patch embeddings)
  - 🟡 P18 (2) learn and add positional embeddings to the inputs at the beginning of each layer;
  - 🟡 P18 (3) add a learned positional embeddings to the inputs at the beginning of each layer (shared between layers)
  - **Food for thought**: Adding positional embeddings to each layer should be important for tasks where preserving spatial information is important, for example, segmentation, object detection etc.

- *Axial Attention*:
  - 🟡 P19 The general idea of axial attention is to perform multiple attention operations, each along a single axis of the input tensor, instead of applying 1-dimensional attention to the flattened version of the input. In axial attention, each attention mixes information along a particular axis, while keeping information along the other axes independent.
  - *Axial-ViT*: 🟡 P19 ViT to process inputs in the 2-dimensional shape, instead of a 1dimensional sequence of patches, and incorporate Axial Transformer blocks, in which instead of a self-attention followed by an MLP, we have a a row-self-attention plus an MLP followed by a column-self-attention plus an MLP.
  - *Result*: 🟡 P19 , both Axial-ViT-B/32 and Axial-ViT-B/16 do better than their ViT-B counterpart in terms of performance, but it comes at 🟡 P20 the cost of more compute. This is because in Axial-ViT models, each Transformer block with global self-attention is replaced by two Axial Transformer blocks, one with row and one with column selfattention and although the sequence length that self-attention operates on is smaller in axial case, there is a extra MLP per Axial-ViT block.

- **Food for thought (CNNs vs ViTs)**:
  - With transformers you get the benefit that you don't downsample. You always maintain the high resolution just like HRNet. But do you have hierarchical aggregation (small scale and large scale) in transformers like CNNs?
  - Max-pooling in CNNs summarize the outputs of a neighbouring groups of neuron in the same kernel map. How to do something similar in transformers? Or, is it even required/necessary in transformers?
  - The intuition of multiple heads in multi-head attention is to learn independent concepts/feature in each head. How do we ensure that there is no redundancy between heads i.e. each head learns different concepts? Also, is multi-head attention similar to group convolutions?