# Vision Transformers

By Shivani Bhatia

→ Transformer architecture for interpreting & understanding images, known as Vision Transformers (ViTs).

→ ViTs divide images into patches, processing them like words in a sentence, to enhance visual data understanding.

→ ViTs excel in object detection, image classification, segmentation, action recognition, generative modelling & multimodal tasks.

## # Transformer Recap

→ Transformer module comprises of

several key layers :

(i) **Self-attention layers :** Layer helps in capturing the relationship between different words (tokens) in a sentence.

(ii) **Normalisation layers :** This layer stabilises the learning process & improves convergence.

(iii) **Feed forward layers :** This layer processes the normalised output further to produce the final embeddings.

The input to the transformer module comprises word tokens that are first converted into word embeddings. These embeddings are then enriched with positional encodings to retain the order

of words in a sentence, which are then fed into transformer encoder.

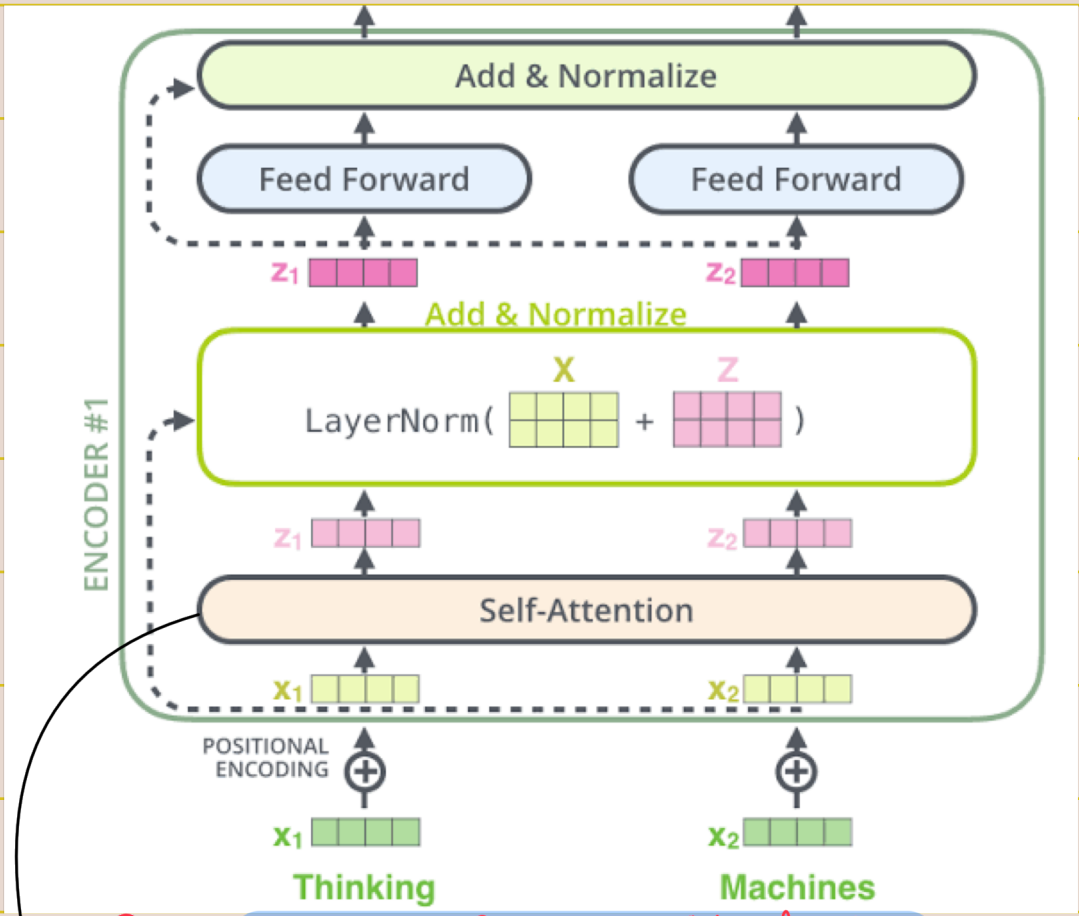



Fig: Transformer Module

# Self-attention mechanism :

⇒ Central to transformer's functioning.

⇒ Involves three learnable matrices :

(i) Query embeddings (Q)

(ii) Key embeddings (K)

(iii) Value embeddings (v)

Goal here is to derive enhanced input representation of input tokens. Achieved by computing a weighted sum of value vectors, where the weights are determined by dot product of the query & key embeddings.

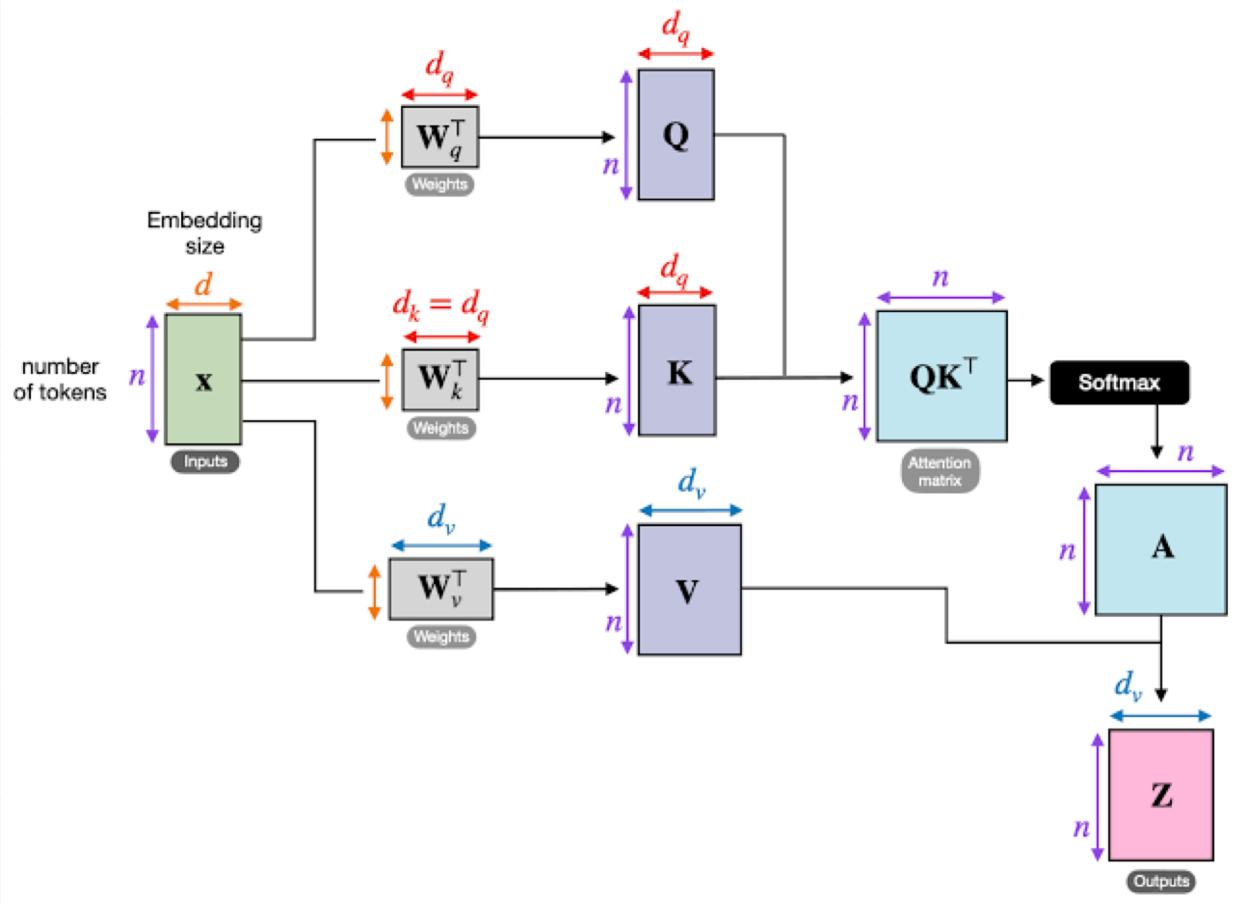
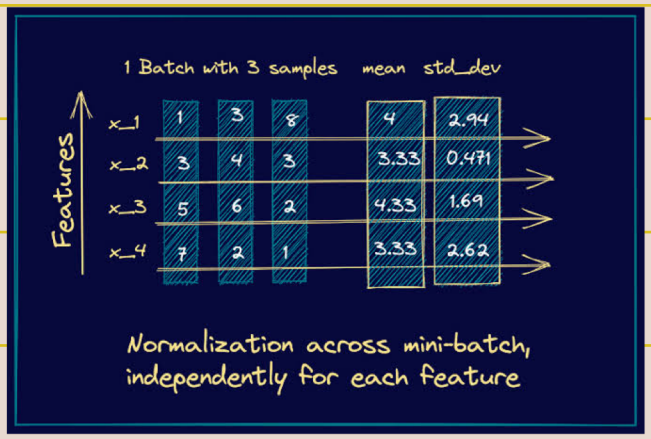
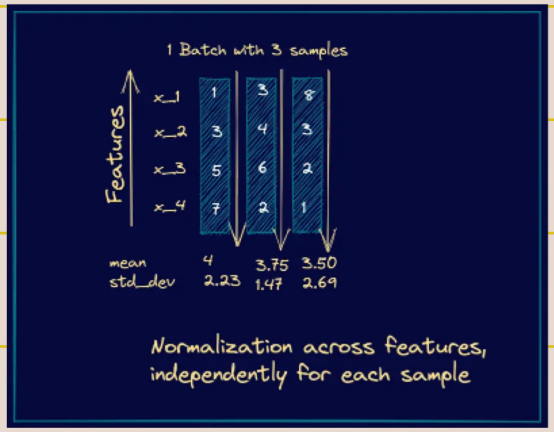



Fig - Self-Attention* → Explaination ahead

| Batch Normalization | Layer Normalization |
| --- | --- |
| (i) Normalises across each feature dimension within a batch. | Normalises across each feature dimension for individual samples independently. |
| (ii) Consider batch dimension | Doesn't consider batch normalisation |
| (iii) Designed for fixed length input sequence within a batch. | Accomadates variable-length input sequence within a batch. |
| (iv) Mean & Std. dev. are computed for each dimension by averaging across samples in mini-batch. | The mean and std. dev. are computed for each feature dimension with single sample. |

(v)




** Layer normalisation is preferred in transformers because it effectively handles variable-length inputs within a single batch.

→ Self-attention* process summarised below:-

(i) Input :- Start with n tokens, each represented as a vector of dimension d.

(ii) Transformation :- These tokens are converted into three sets of embeddings: n query tokens

and n Key embeddings, both with dimensions $d_q$, and n value embeddings with dimension $d_v$.

(iii) Output :- Each transformed into more meaningful representation, which is weighted sum of the value embeddings. The weights are calculated using matrix A, which is derived from dot product b/w query & Key embedding

→ Classification (CLS) tokens are special tokens added to beginning of input sentence in transformers.

→ CLS tokens aggregates the features from all image patches into single embedding, i.e used for final classification.

→ CLS tokens acts as pooling mechanism, aggregating features of all image patches into single representation.

** Self attention layer in ViTs captures relationship b/w patches, similar to words in NLP.

# Architecture of Vision Transformer

→ ViT architecture leverages a transformer encoder, similar to those in language processing, comprising multi head attention, normalisation & MLP layers.

→ ViT architechture is notably similar to architecture of encoder used in language processing.

→ The core concepts of the ViT encoder include multihead attention, a normalisation layer and an MLP.

→ However, key difference lies in method of feeding input to transformer encoder module.

→ Preview of how inputs are processed in ViTs :-

(i) Image tokenization :- Splitting images into patches, reducing complexity & capturing local structures.

(ii) Flattening and embedding patches :- Patches are flattenned & transformed using linear embeddings.

(iii) Positional embedding : Positional embeddings are added to maintain spatial information in sequence-agonistic transformers.

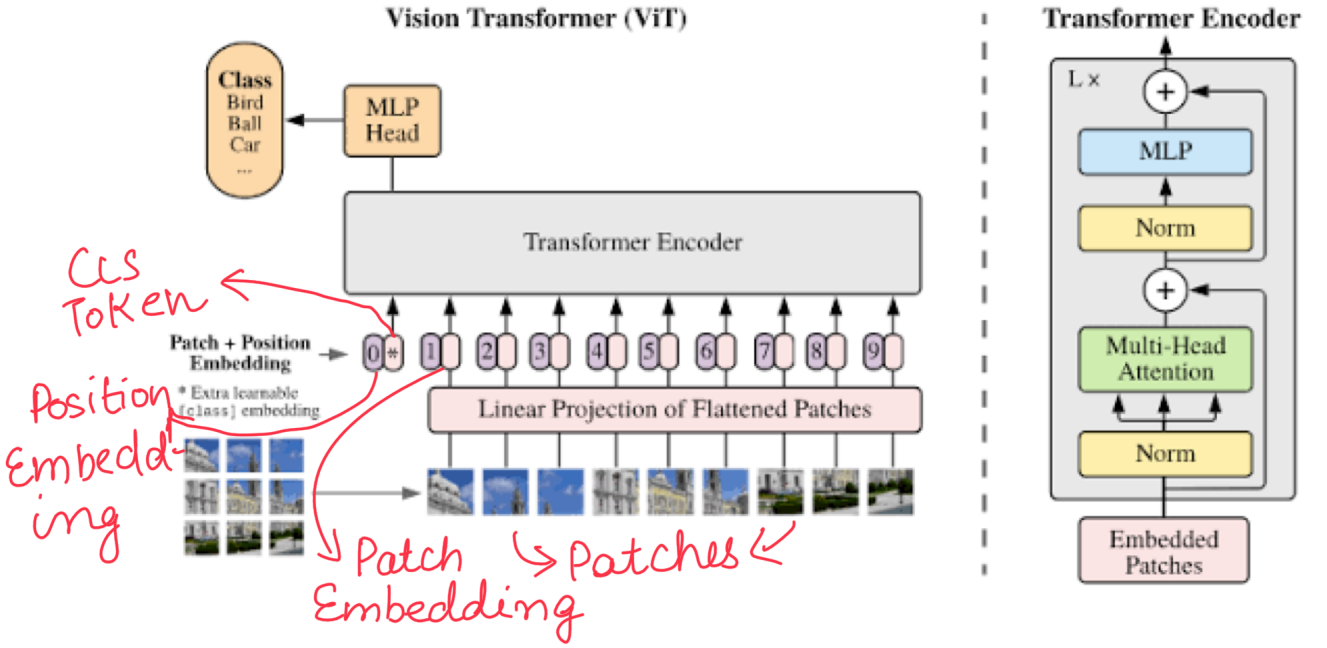



Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification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

# Patch creation process :

* Image is represented as h x w x c.
  - c → Channel No.
  - w → width
  - h → height

(i) An image of size (224 X 224 X 3), let's assume is divided in small patch of size (16 X 16 X 3). Each patch represent small section of

(ii) Each 16 X 16 patch is then flattened into a single vector of size 768 (converting 3D patch into a 1D vector).

(iii) To create final representation, total no. of patches (n) is calculated by dividing total image area by area of one patch.

$$n = hw/p^2$$

$$n = \frac{224 \times 224}{16 \times 16} = 196$$

So, total no. of patches, image will be divided would be 196.

→ Each patch will be represented as a 768-dimensional vector & these vectors are then combined to form a 196 X 768 matrix, with each row corresponding to one of flattened patch vectors.

→ These vectors undergo linear projections, creating patch embeddings for transformer input.

# Image flattening & Positional Encoding

→ The patches which are created are flattened into one-dimensional vectors of size (1 X 768).

→ To feed these vectors into the transformer, an intial input representation $z^0$ is

constructed.

Patch Embedding

$$z_0 = [x_{class}; x^1_p E; x^2_p E \cdots x^N_p E] + E_{pos}$$

Class Token

Position Embedding

→ The embeddings are organised into two matrices - the patch embedding E, which includes the CLS token followed by embeddings for each patch, and the position embedding matrix (Epos), which includes embeddings for positions from 0 to n, where n is total no. of patches (tokens) in the input image.

→ The dimensions of these matrices are crucial. The patch embedding matrix E is sized $p^2 * c * d$, where $p^2 * c$ is dimensionality of a single patch & d is desired output di-

mension.

→ For example, with d=64, each patch embedding vector would be 1X64.

→ Position Embedding matrix, Epos is sized (n+1)*d → dimensions
↳ CLS token + n tokens

→ Once embedded, these patches are fed into transformer encoder, which applies multiple layers of transformations.

→ Each layer processes the input embeddings & outputs the transformed embeddings, while maintaining the same no. of outputs & inputs.

→ Key aspect in transformer-encoder step is

that CLS token undergo global attention across all input patches.

→ This allows CLS token to capture comprehensive representation of the entire image, without bias from specific patches. This enhances decision making processes, such as image-classification.

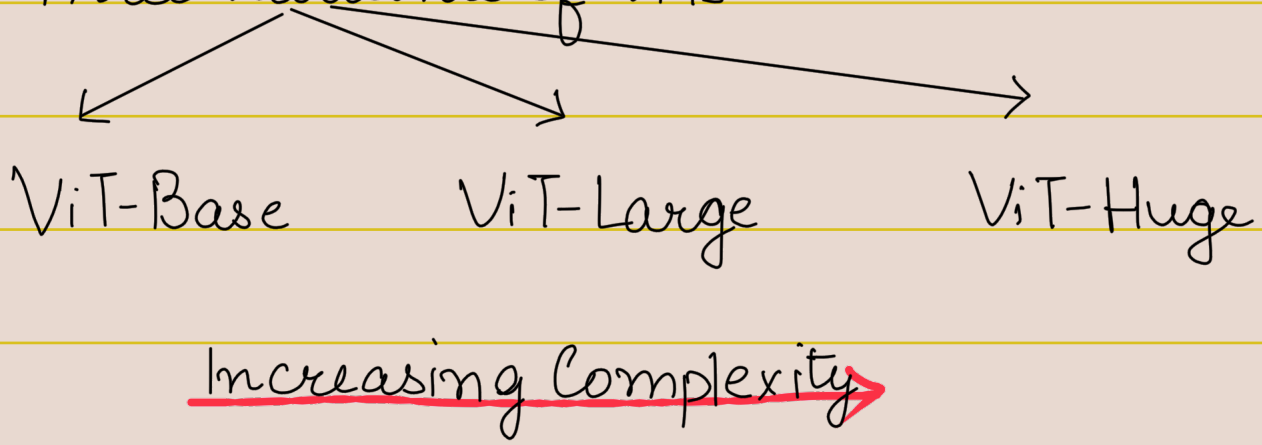
→ From transformer encoder, MLP head is attached to CLS token representation. The MLP head outputs probabilities across different classes, akin to standard neural network classification.

→ In CNN, batch normalization is used to speed up training while in ViTs, layer normalization is used following design principles of

language transformers.

# Types of ViTs

→ Three variants of ViTs

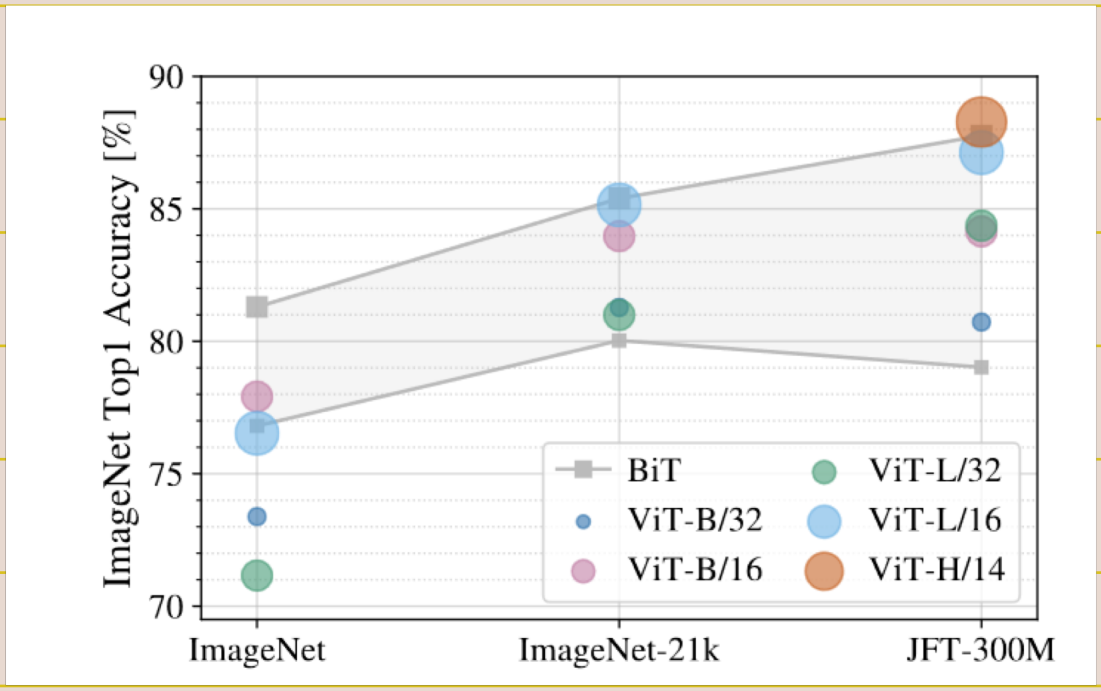



| Model | Layers | Hidden Size D | MLP Size | Heads | Params |
|---|---|---|---|---|---|
| ViT – Base | 12 | 768 | 3072 | 12 | 86M |
| ViT-Large | 24 | 1024 | 4096 | 16 | 307M |
| ViT – Huge | 32 | 1280 | 5120 | 16 | 632M |

→ The performance of ViT is lower than that of CNN's when trained on ImageNet, which has limited training data.

→ As data size increases ViT, particularly the huge variant with 14 patch size, begin to outperform CNN's.

→ On the JFT-300M data set, ViT achieved higher classification accuracy than best performing CNN.



ViTs may not match CNN's performance on smaller data sets but can surpass it when provided with ample training data.

** Further explore ViT Python implementation & Google Vit.