An Overview of Transformers

1. Transformers
2. Vision Transformers (ViTs)
3. Video Vision Transformers

1. Transformers

• Transformer architectures were first introduced in [Vaswani, 2017*], with "Attention is All You Need." Achieved new state of the art in NLP (machine translation using BLEU score), requiring only a fraction of time of previous SOTA models to train.

• Quickly, Transformers displaced RNN and LSTM architectures as the *de facto* NLP DL models; more recently, Transformers have displaced CNNs in CV.

• Basic high-level advantages of Transformers over RNN-based models: (1) Transformers can efficiently learn **long-range semantic dependencies**; (2) Transformer training is **parallelizable** (on the token level), whereas RNN-based models are inherently sequential.

* Vaswani et al., "Attention is All You Need," NIPS 2017: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

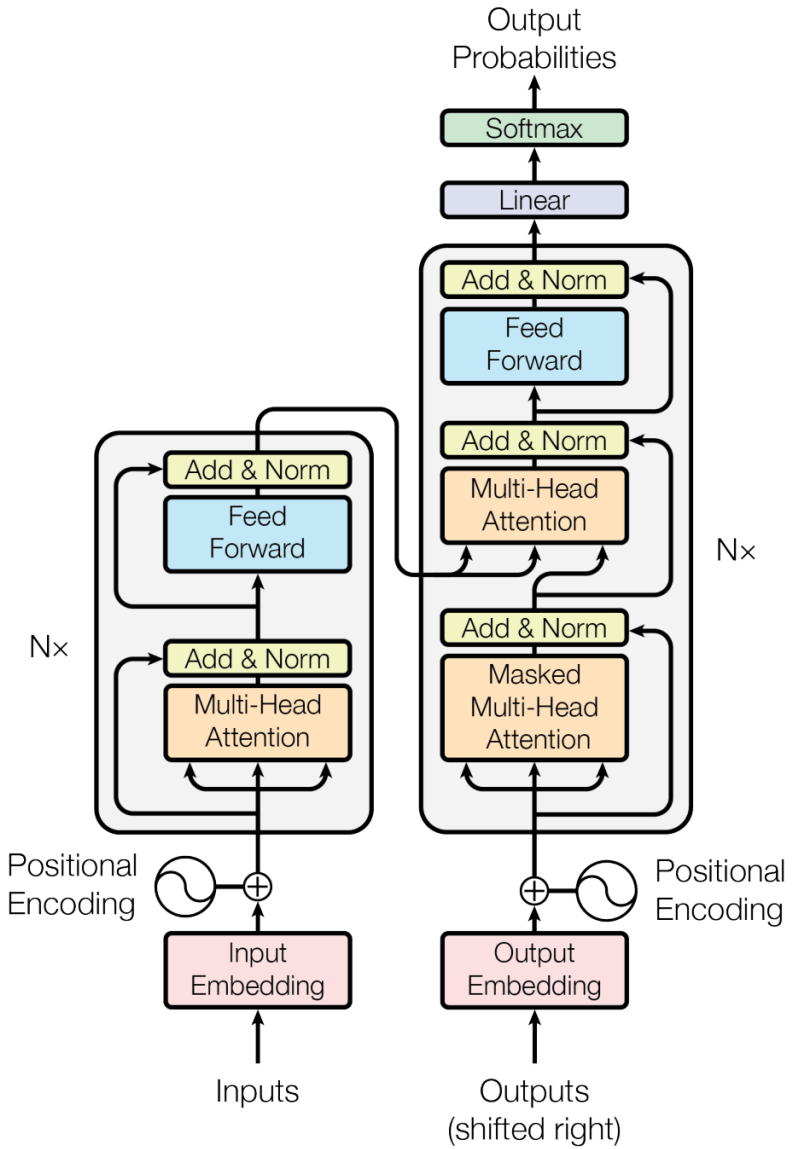
Intel Labs: Anthony Rhodes

1. Transformers

• Let's dive into the nuts and bolts of the original Transformer model.

• Two macro components: (1) **Encoder** (left) and (2**) Decoder** (right).

• Several components of Transformers are standard, including Feed Forward module, Layer Norm, linear project layer, softmax, etc.



Intel Labs: Anthony Rhodes

1. Transformers

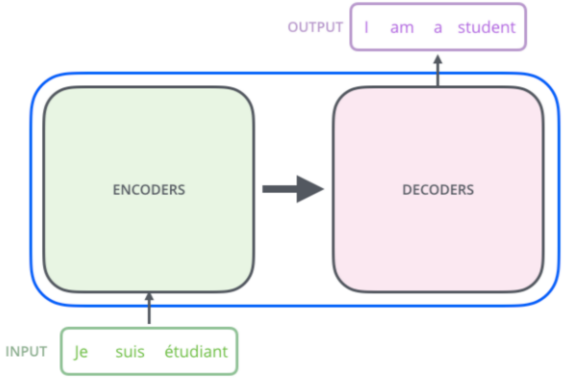
• Let's dive into the nuts and bolts of the original Transformer model.

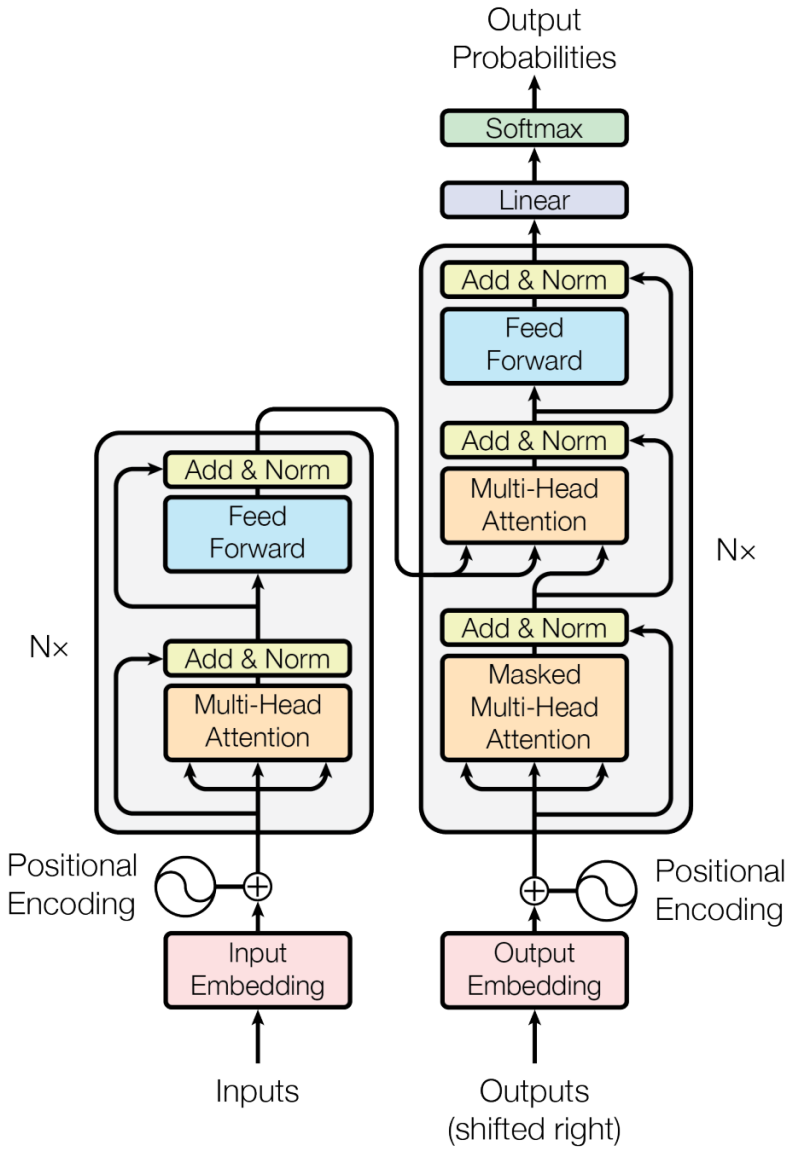
• Two macro components: (1) **Encoder** (left) and (2**) Decoder** (right).

• Several components of Transformers are standard, including Feed Forward module, Layer Norm, linear project layer, softmax, etc.

• So, what's new? **Multi-Head Attention** (and Masked Multi-Head Attention); recall: attention is all you need.

• **Encoder**: encodes (in NLP application) first language text (e.g., French) into latent code; **Decoder**: decodes latent code into second language text (e.g., English).



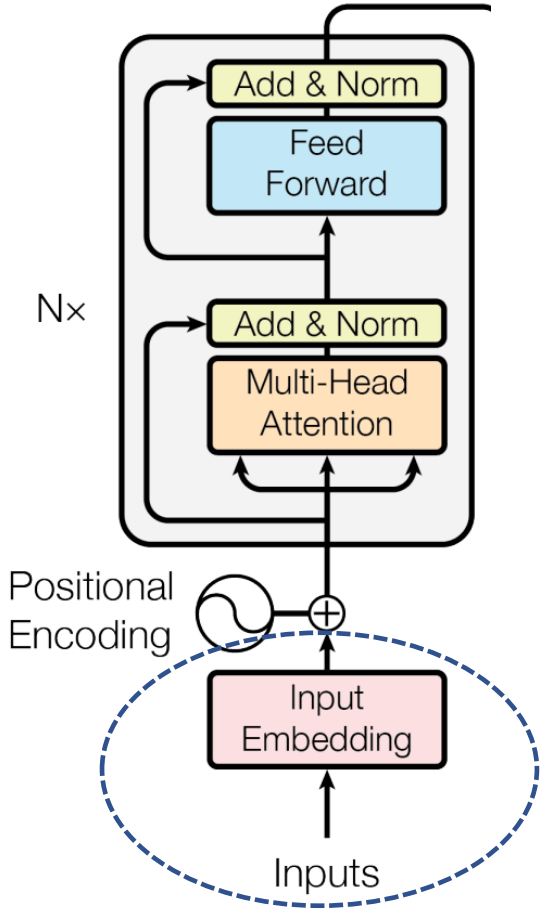


Encoder Structure

• **Encoder**: encodes (in NLP application) first language text (e.g., French) into latent code.

Input: Encoder takes as input a (usually 1-hot, etc.) representation of individual words that make up a sentence/paragraph.

• Each word is then projected linearly via an **input embedding** (note this can be done in parallel for each word). This yields an input **embedding matrix** $\boldsymbol{\grave{X}}$ of dimension (#words, embedding dim – e.g., 512).

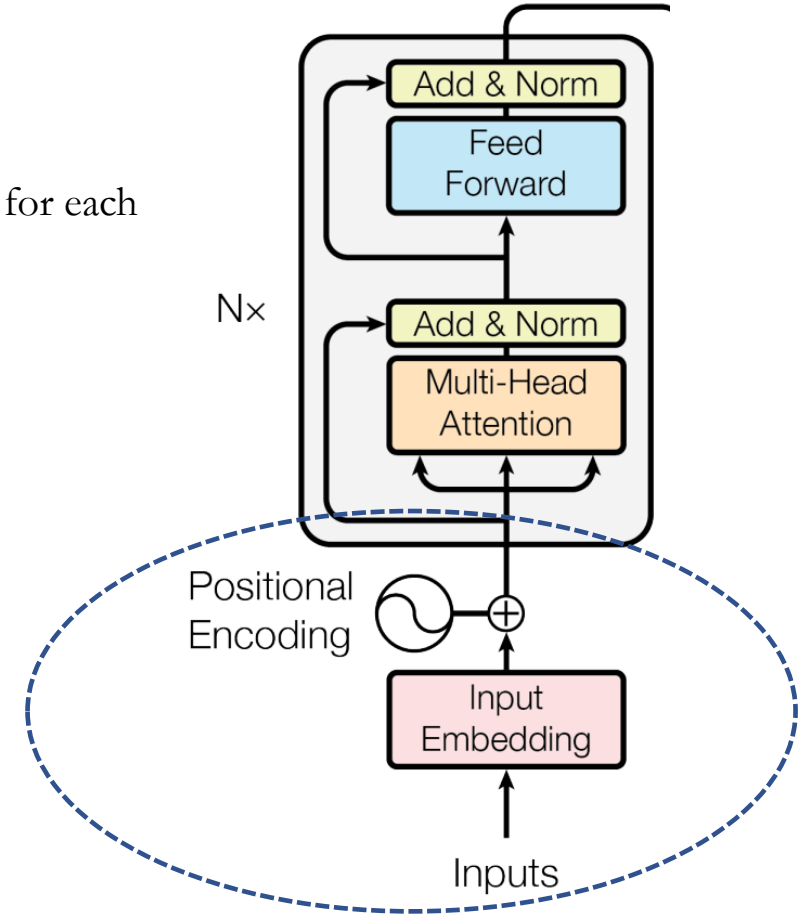


Encoder Structure

• Each word is then projected linearly via an **input embedding** (note this can be done in parallel for each word). This yields an input **embedding matrix** $\boldsymbol{\grave{X}}$ of dimension (#words, embedding dim).

• Next, positional encodings are added to the embedding matrix:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Where *pos* denotes the word position in the input sentence, and *i* represents dimension component; d_{model} denotes the embedding dimension. Denote the input after positional encoding as $\boldsymbol{X}$.

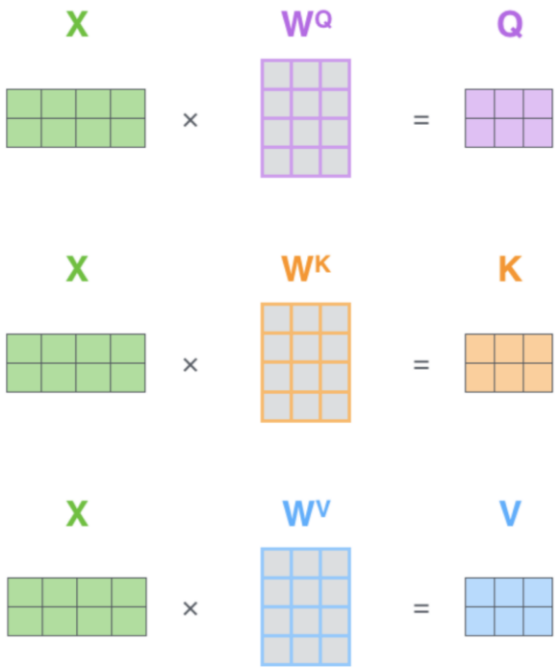
*Note that the positional encoding is required because the Transformer is recurrence and convolution free. In other words, some information about the absolute position of the tokens must be included. The Transformer learns a **disentangled representation** of the input.

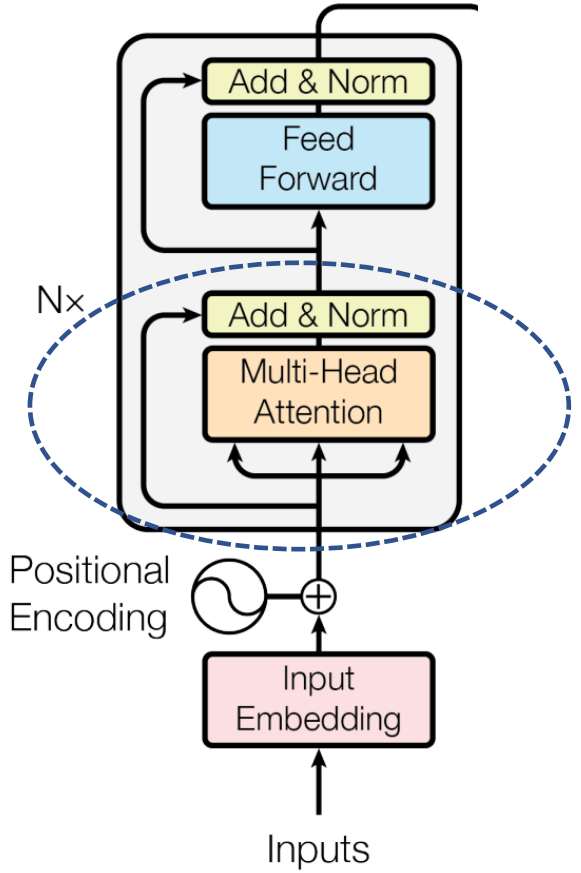
*Recent Transformers have shown that **making the positional encoding learnable** can further improve model performance.

Encoder Structure

• Self-Attention Mechanism: Self-attention (SA) represents the key architectural innovation in Transformers. SA allows the Transformer to learn meaningful correlations between the input tokens.

• This technique is inspired by approaches from text mining. For each SA component, we map $\boldsymbol{X}$ into three new matrices: $\boldsymbol{Q}$ (query), $\boldsymbol{K}$ (key) and $\boldsymbol{V}$ (value) via multiplication with learnable weight matrices: W_Q, W_K, and W_V , respectively.

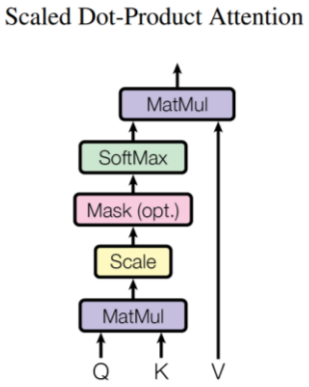


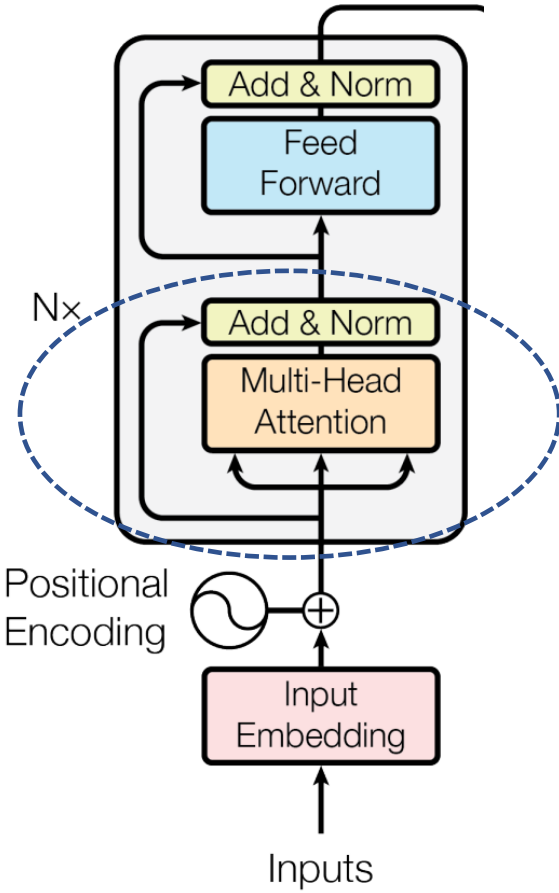


Encoder Structure

• This technique is inspired by approaches from text mining. For each SA component, we map $\boldsymbol{X}$ into three new matrices: $\boldsymbol{Q}$ (query), $\boldsymbol{K}$ (key) and $\boldsymbol{V}$ (value)

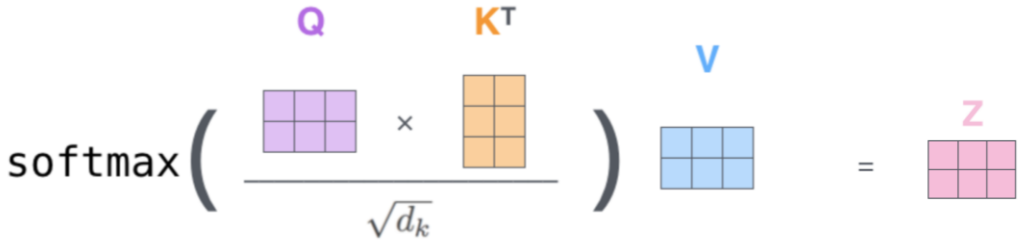
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$





Where QK^T yields a similarity matrix between query and key tokens; division by $\sqrt{d_k}$ (the dimension of the key matrix) normalizes this matrix and leads to more stable gradient values at training time.

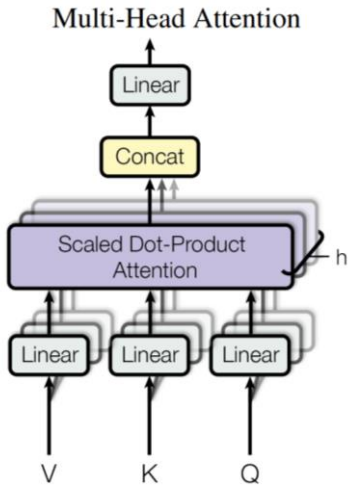
• The softmax transformation renders a probability distribution per row, i.e. the **correlation score** of a query word (row i) with each key word (column j). Finally, this row-normalized matrix is multiplied by the Value matrix, yielding a weighted sum of each row of the normalized correlation scores (the weights in V can connote word importance, for instance).

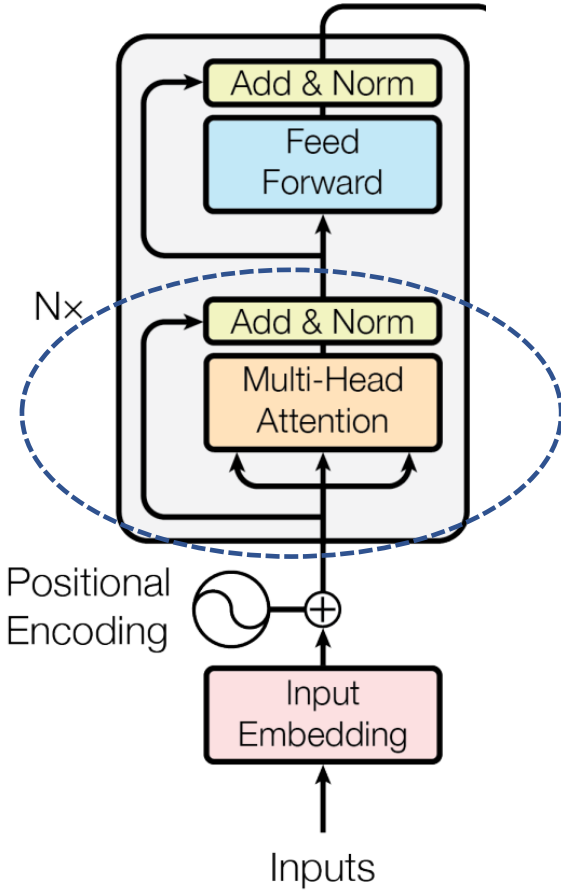


Encoder Structure

• **Multi-Headed Attention** (MHA) indicates that the previous processes are repeated, independently, for h (e.g., $h = 8$) Attention Heads. The latent matrices produced by each head are concatenated together; an additional (tunable) weight matrix W_O is used to project this concatenation. The MHA is intended to improve the expressivity of the model, including disambiguating different semantic uses of the same word.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

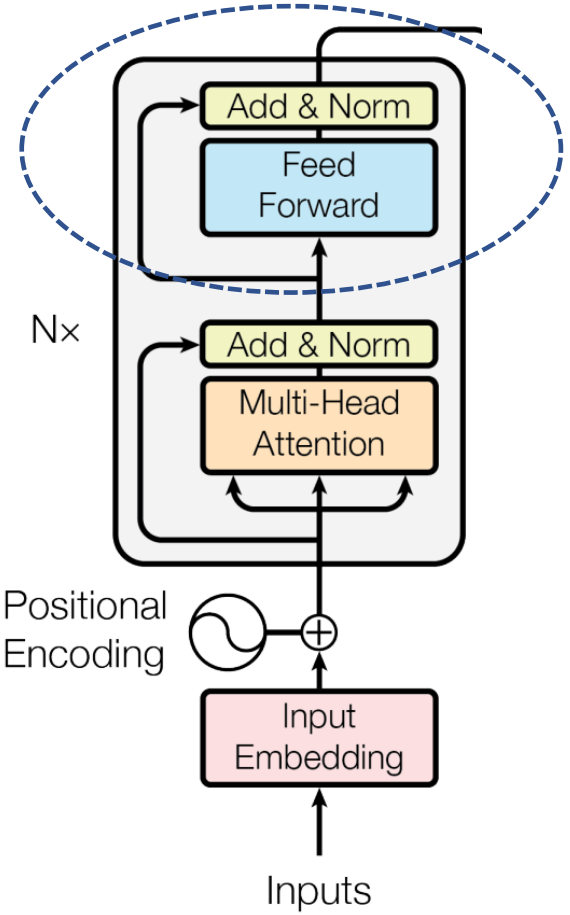


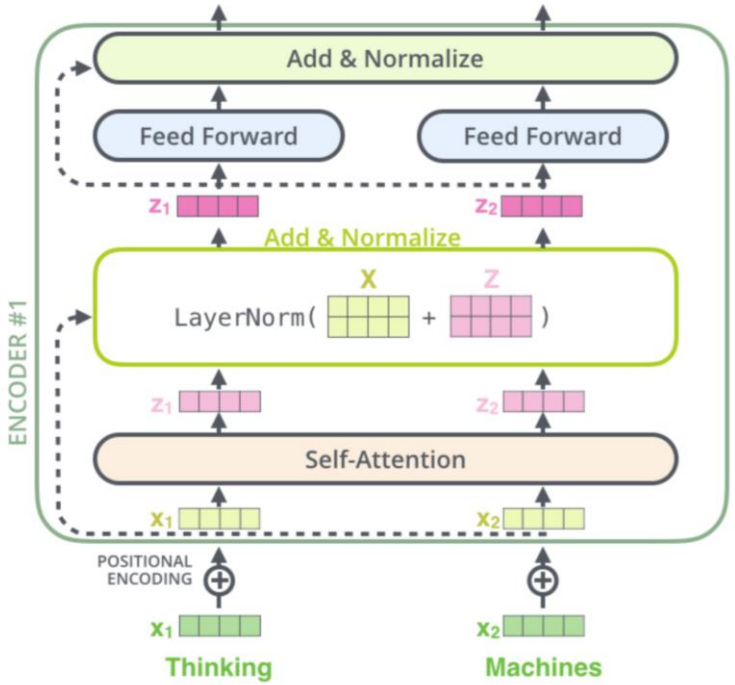


Encoder Structure

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

• Next, the output of the MHA is added to ***X*** via residual connection, followed by **Layer Normalization**. Lastly, this feature representation is passed through a **single layer feed-forward network**, with residual connection. This basic series of network blocks is duplicated exactly N times (e.g., N = 6).

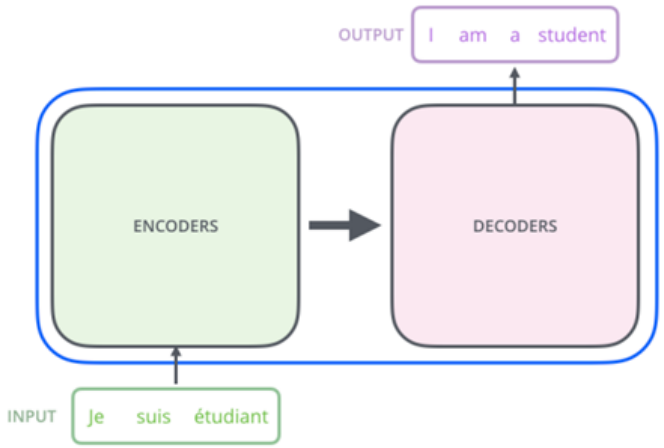


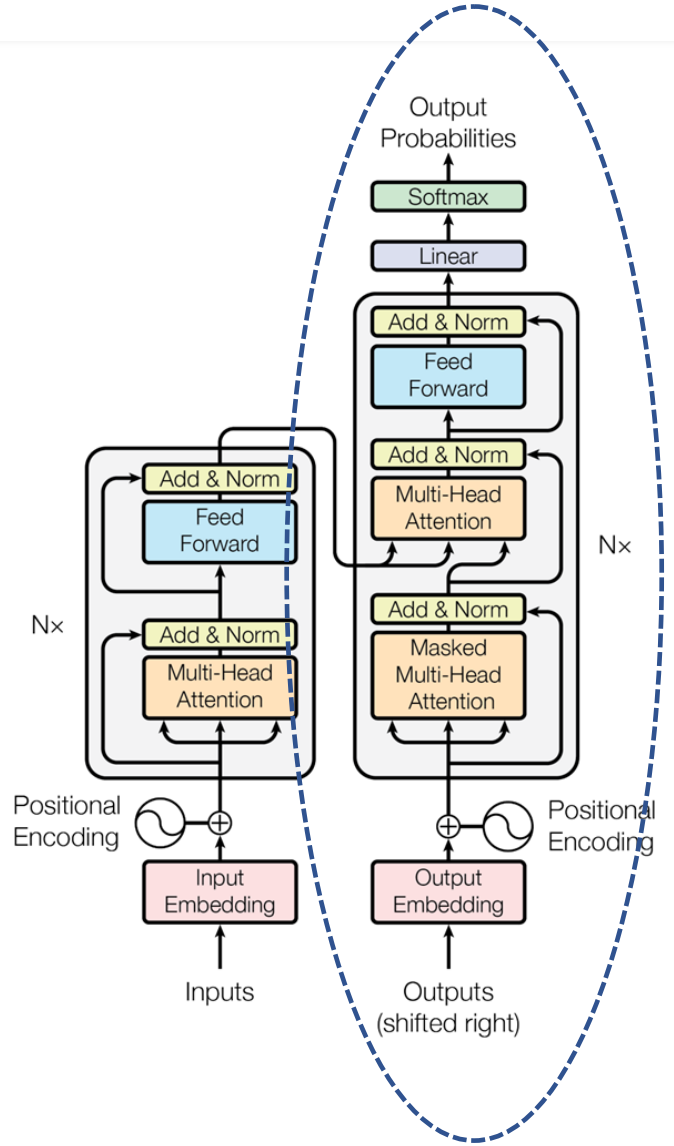


Decoder Structure

• The Transformer Decoder operates in largely the same way as the Encoder, as it includes positional encoding, MHSA modules followed by residual connections and Layer Normalization, as well as a single Layer feed-forward NN.

• Recall that the Decoder is tasked with transforming the latent representation generated by the Encoder into (in the case of machine translation) the equivalent sentence in another language.





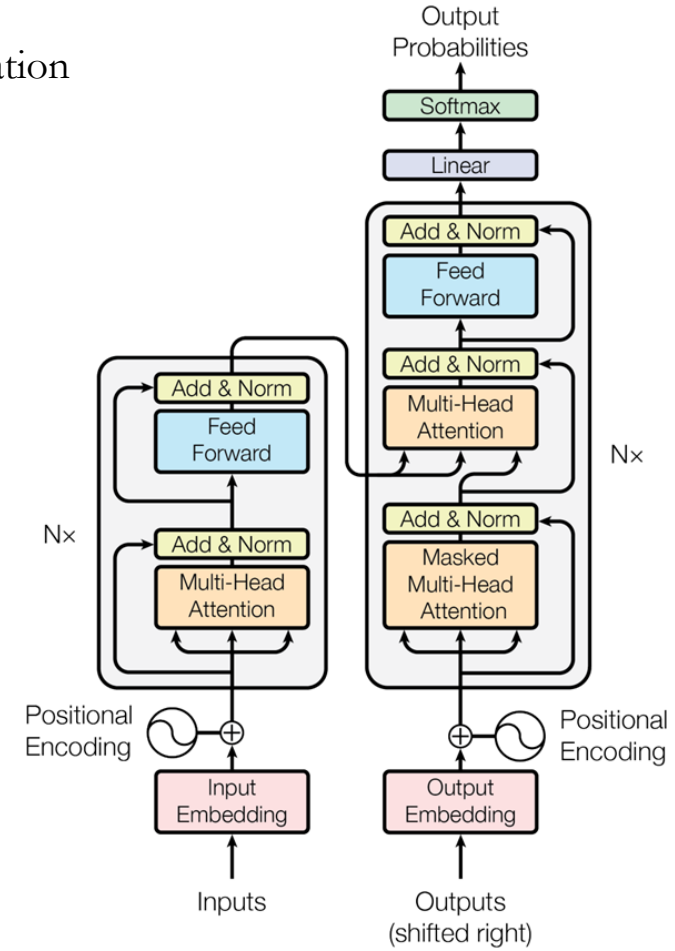
Decoder Structure

• So, what then is different about the Decoder? More concretely, the Decoder receives the latent representation of the input produced by the Encoder as a residual connection (see diagram).

• When the latent representation is passed through the Decoder it elicits a distribution of probabilities over the output language dictionary, i.e. it outputs a word (take the max of the predictions).

• Next, this output word is then passed through the Decoder, where it passes through a **Masked Multi-Head Attention** module.

• This network component is identical to MHA from before, except that at training time, the subsequent words in the translation sentence are "masked", i.e., hidden from the Decoder, so that it does not have access to the ground-truth label during training.

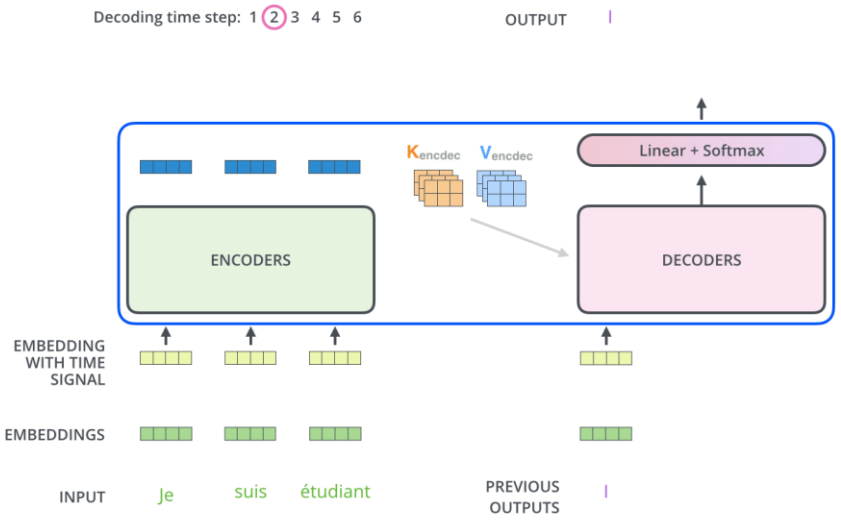


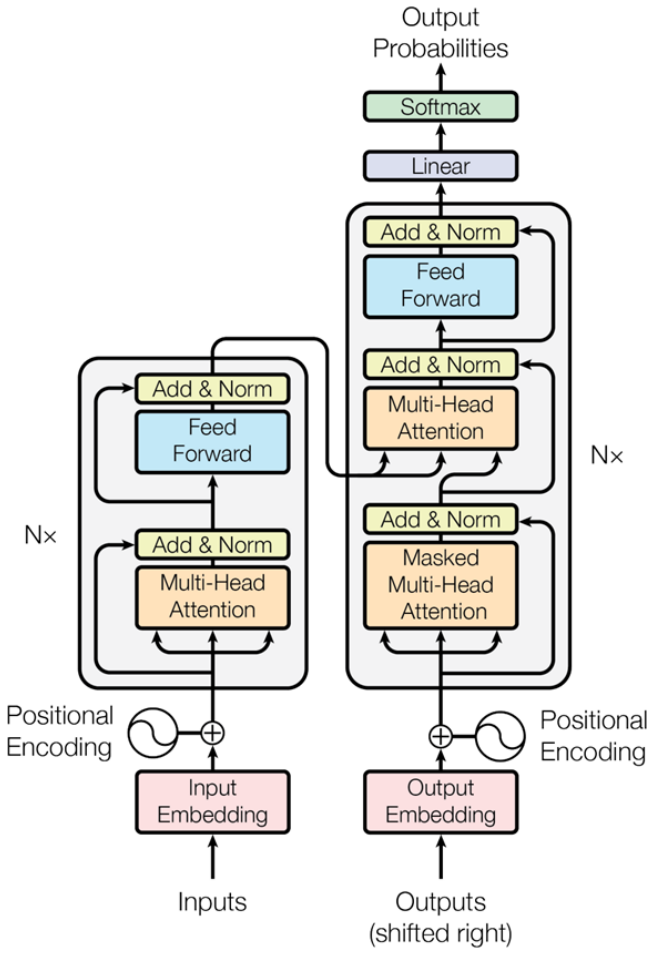
Decoder Structure

• Next, this output word is then passed through the Decoder, where it passes through a **Masked Mult-Head Attention** module. This network component is identical to MHA from before, except that at training time, the subsequent words in the translation sentence are "masked" (i.e. hidden) from the Decoder, so that it does not have access to the ground-truth label during training.

• As in the previous step, the residual from the Encoder is passed to the MHA (now attention is calculated between the input sentence and the translation sentence generated thus far).

• At the end of this sequential process (each word in the translated sentence is generated one at a time) the Decoder generates a special <end> token to signify the end of the translation.

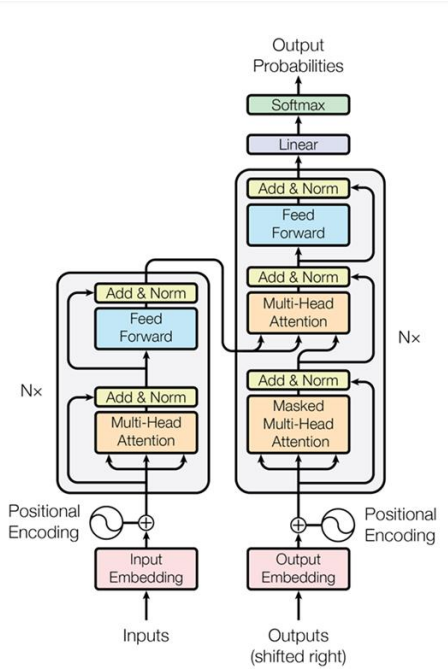




Transformer Summary

(+) New SOTA!

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	



(+) Can learn **long-range semantic dependencies**.
(+) Training is **parallelizable** (unlike RNN-based models which are inherently sequential), meaning training FLOPS are relatively small.
(+) Model performance **scales very well with # of parameters**!
(+) Much like previous NLP vector space models (e.g., word2vec), **can use vanilla Transformer as pre-text model**, and then fine-tune for specific downstream task (e.g., query-answer, sentiment analysis, captioning, chatbot, etc.).

(-) Need **colossal amount of data** for large Transformers (think GPT-3).

• With BERT (Devlin et al., Google, 2019), the authors proposed (2) unsupervised, NLP-based pretext tasks for large-scale Transformer pretraining:

(1) Masked Language Model (MLM) – where some percentage of input tokens are randomly masked, and the model predicts these masked tokens.

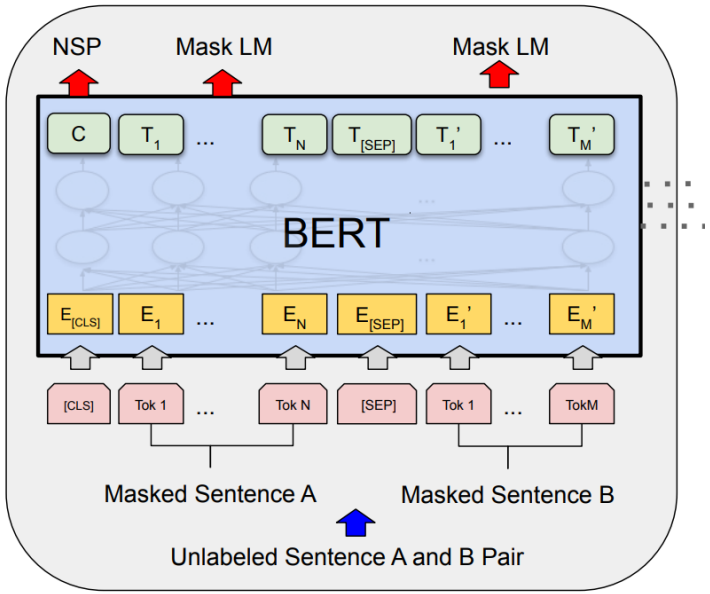
(2) Next Sentence Prediction (NSP) – where sentence relationships are learned using a binarized sentence prediction, i.e., given sentences: A|B, does B follow A (50% of the time during training B does follow A in the training corpus).

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

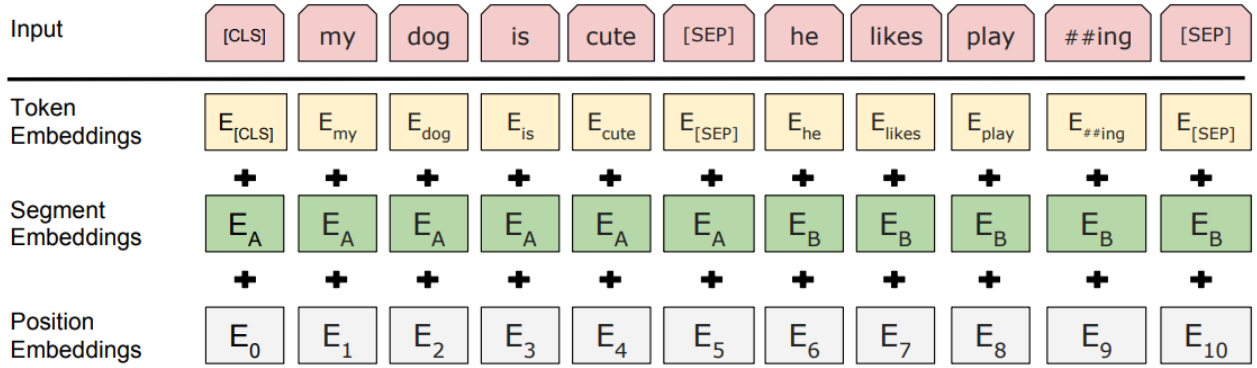
• With BERT, the authors proposed (2) unsupervised, NLP-based pretext tasks for large-scale Transformer pretraining:

(1) **Masked Language Model** (MLM) – where some percentage of input tokens are randomly masked, and the model predicts these masked tokens.

(2) **Next Sentence Prediction** (NSP) – where sentence relationships are learned using a binarized sentence prediction, i.e. given sentences: A | B, does B follow A (50% of the time during training B does follow A in the training corpus).



Pre-training



An Overview of Transformers

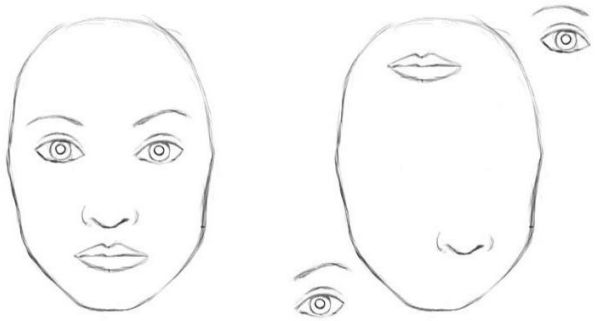
1. Transformers
2. Vision Transformers (ViTs)
3. Video Vision Transformers

Limitations of CNNs

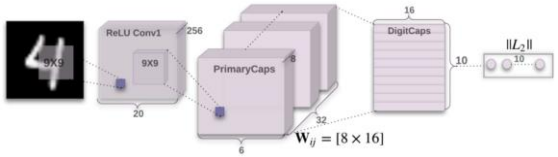
•According to Geoffrey Hinton*, it is unfortunate in some sense that CNNs work so well, because they have serious flaws which he believes "will be hard to get rid of."

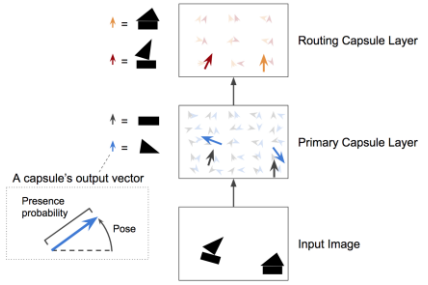
These flaws include (according to Hinton):

- **Inefficiencies in backpropagation** paradigm itself
- **Poor translational invariance**
- **Lack of "pose" information** – absence of nuanced information about relative position and orientation of parts of an object. These is sometimes referred to as the "Picasso problem."



- Hinton proposed **Capsule Nets**** (2017) to address some of these issues.





*https://www.youtube.com/watch?v=rTawFwUvnLE&t=8s

** https://arxiv.org/abs/1710.09829

Vision Transformers

• The introduction of high-performing Transformers models in NLP (2017) and Computer Vision (2020) encapsulates an effort to replace hand-written features or inductive biases with general-purpose neural architectures powered by data-driven training.

• In CV in particular, SOTA **Vision Transformers are thought to benefit from the lack of strong inductive biases** exhibited by traditional CNN models, including inherently *spatial*, *local* and *hierarchical* feature processing operations.*

Vision Transformers

• The introduction of high-performing Transformers models in NLP (2017) and Computer Vision (2020) encapsulates an effort to replace hand-written features or inductive biases with general-purpose neural architectures powered by data-driven training.

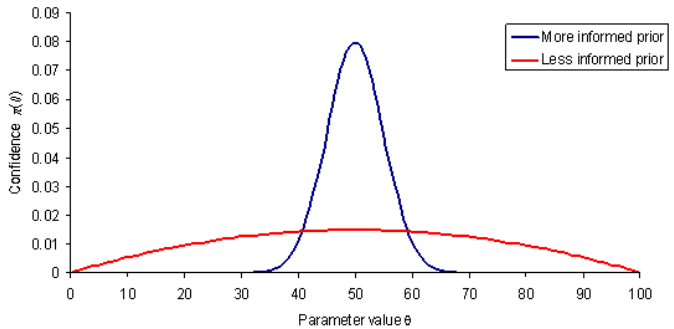
• In CV in particular, SOTA **Vision Transformers are thought to benefit from the lack of strong inductive biases** exhibited by traditional CNN models, including inherently *spatial*, *local* and *hierarchical* feature processing operations.*

• The **absence of many of these convolution-like inductive biases** can lead to improved generalizability; on the other hand, the lack of inductive bias presents unique challenges, as such models can require either large quantities of data to train or specialized optimization functions.*

Likelihood of the data Model Prior

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}$$

Posterior of model, given data



Classical *Bayes' Rule* captures the trade-off between models with high inductive bias and those with low inductive bias (i.e. *informed priors* vs *uniformed priors*). Learning algorithms utilizing models with low inductive bias can learn model types that are otherwise inaccessible via models with high inductive bias.

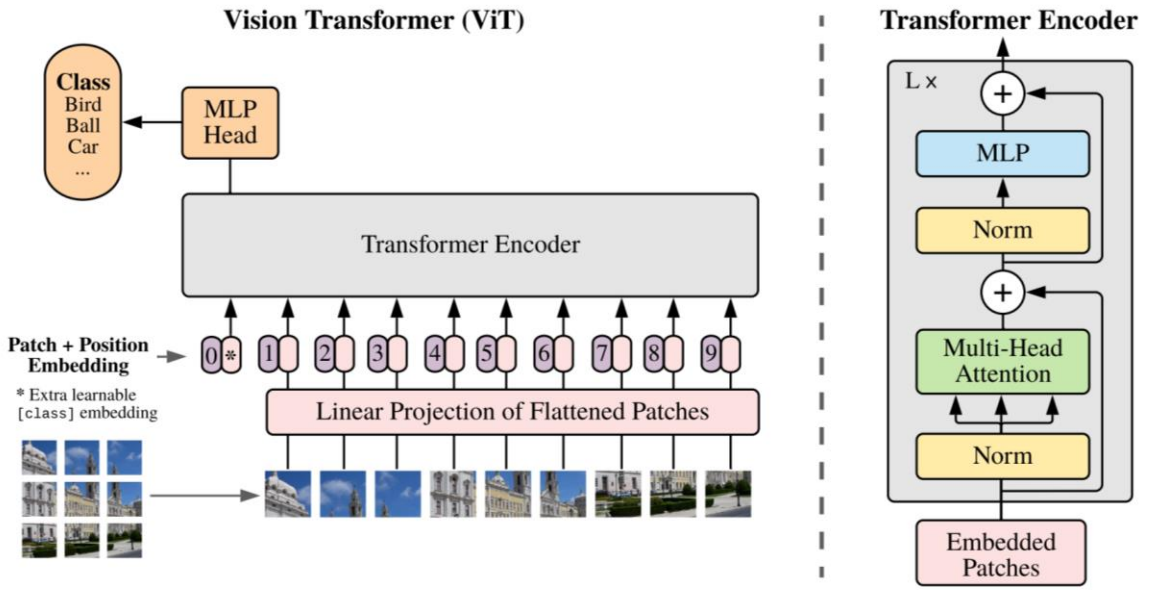
*Chen et al., "When Vision Transformers Outperform ResNets…": https://arxiv.org/pdf/2106.01548.pdf

Vision Transformers

• "An Image is Worth 16x16 Words…" (Dosovitskiy et al., ICLR 2021), a highly-influential pure ViT (no convolutions).

• Fundamentally same architecture as original Transformer, except that 'tokens' here are image patches; MLP is 2-layer. Note that only an Encoder sub-network is used. Notice that **fundamental challenge with ViTs is circumventing ostensible O(n^2) pixel-level attention calculation.**

• For this reason, most subsequent Vision Transformer work focuses on efficient attention/scale computations.



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

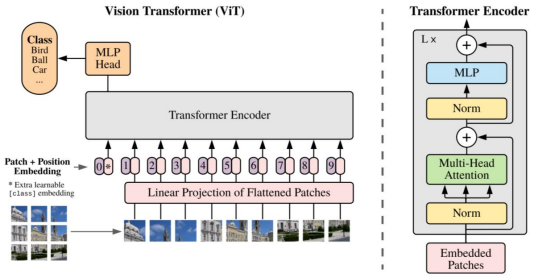
$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$
$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L$$
$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L$$
$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$

* https://arxiv.org/pdf/2010.11929.pdf

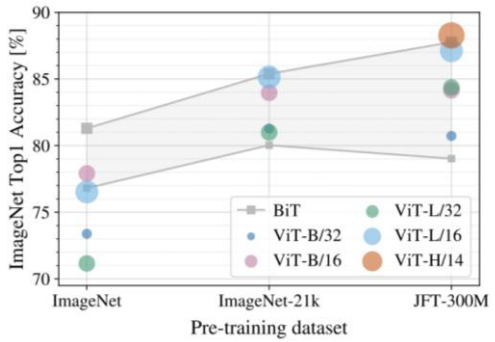
Vision Transformers

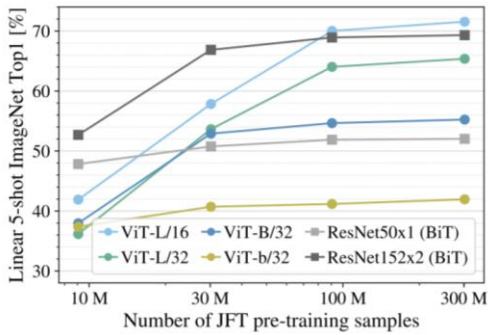
• ViT produces SOTA results – however due to size of model, requires large data sets (it is believed). Note that much less computation is required to train ViT compared to equivalent-sized CNNs.

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



	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k





• Notably, ViT is highly performant – when trained on large datasets. In particular, **when training on large datasets such as ImageNet-21k (~14M images) and JFT (Google proprietary, ~300M images), ViT performs best in class**. However, when trained on smaller datasets, e.g., Imagenet (~1M images) the performance is worse than ResNet.

• These results are fairly intuitive. Because ViT possesses fewer inductive biases than standard CNNs, without more data/regularization, it is likely to underperform.

ViTs vs. CNNs

- "Do Vision Transformers See Like CNNs?" (Dosovitskiy, Google Brain August 2021)

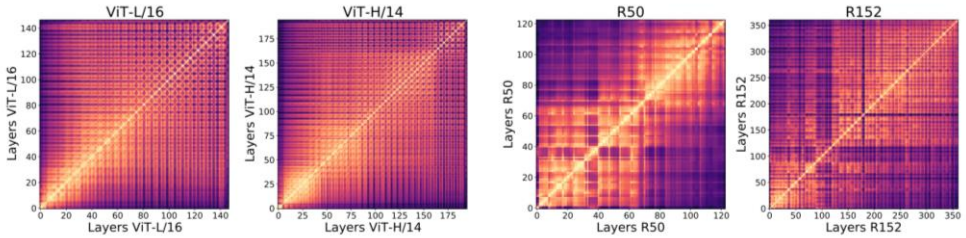


Figure 1: Representation structure of ViTs and convolutional networks show significant differences, with ViTs having highly similar representations throughout the model, while the ResNet models show much lower similarity between lower and higher layers. We plot CKA similarities between all pairs of layers across different model architectures. The results are shown as a heatmap, with the x and y axes indexing the layers from input to output. We observe that ViTs have relatively uniform layer similarity structure, with a clear grid-like pattern and large similarity between lower and higher layers. By contrast, the ResNet models show clear stages in similarity structure, with smaller similarity scores between lower and higher layers.

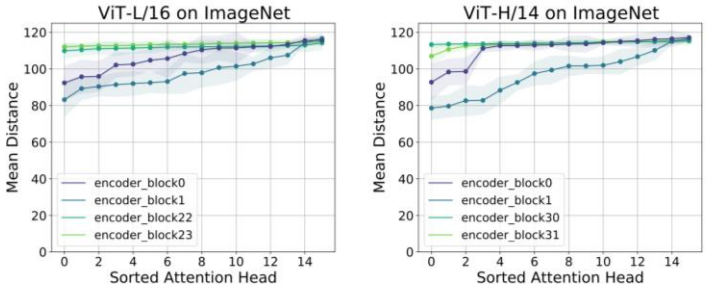


Figure 4: With less training data, lower attention layers do not learn to attend locally. Comparing the results to Figure 3, we see that training only on ImageNet leads to the lower layers not learning to attend more locally. These models also perform much worse when only trained on ImageNet, suggesting that incorporating local features (which is hardcoded into CNNs) may be important for strong performance. (See also Figure C.5.)

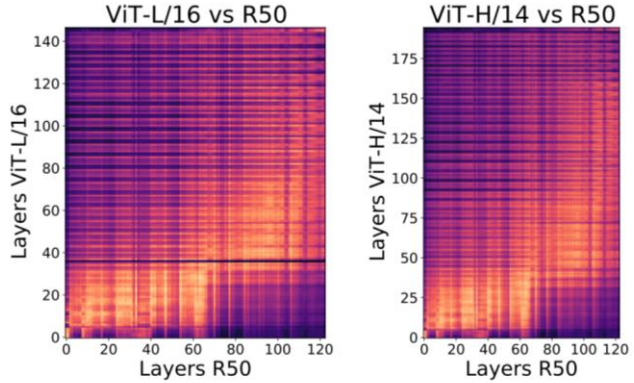
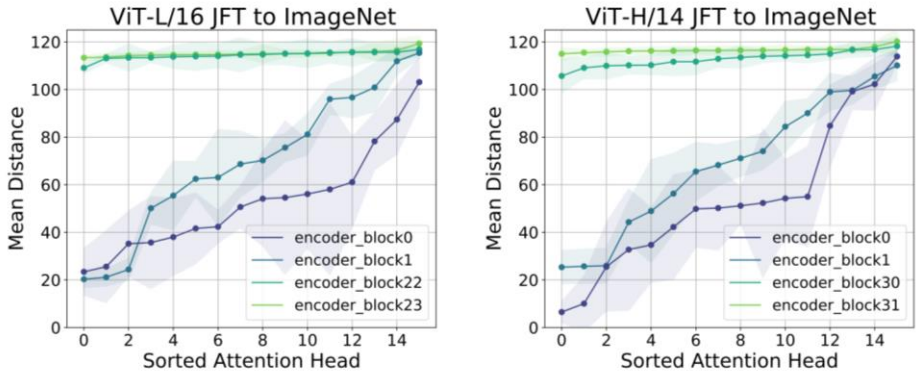
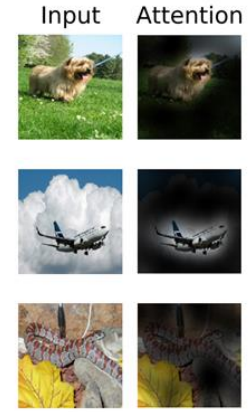


Figure 2: Cross model CKA heatmap between ViT and ResNet illustrate that a larger number of lower layers in the ResNet are similar to a smaller set of the lowest ViT layers. We compute a CKA heatmap comparing all layers of ViT to all layers of ResNet, for two different ViT models. We observe that the lower half of ResNet layers are similar to around the lowest quarter of ViT layers. The remaining half of the ResNet is similar to approximately the next third of ViT layers, with the highest ViT layers dissimilar to lower and higher ResNet layers.





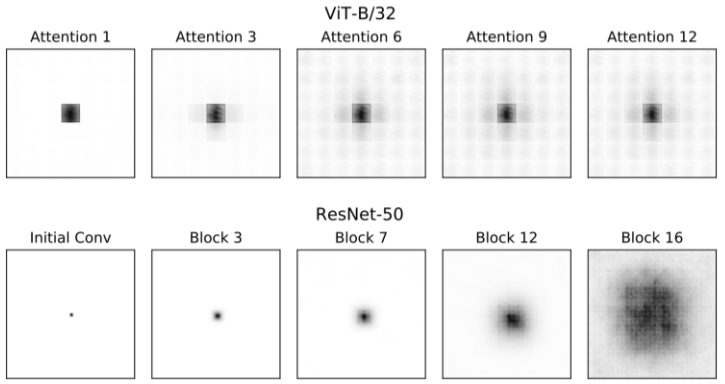


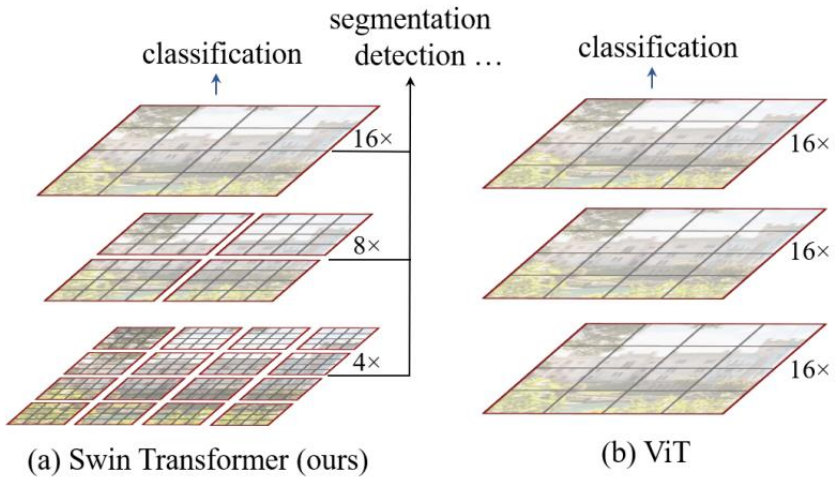
Figure 6: ResNet effective receptive fields are highly local and grow gradually; ViT effective receptive fields shift from local to global. We measure the effective receptive field of different layers as the absolute value of the gradient of the center location of the feature map (taken after residual connections) with respect to the input. Results are averaged across all channels in each map for 32 randomly-selected images.

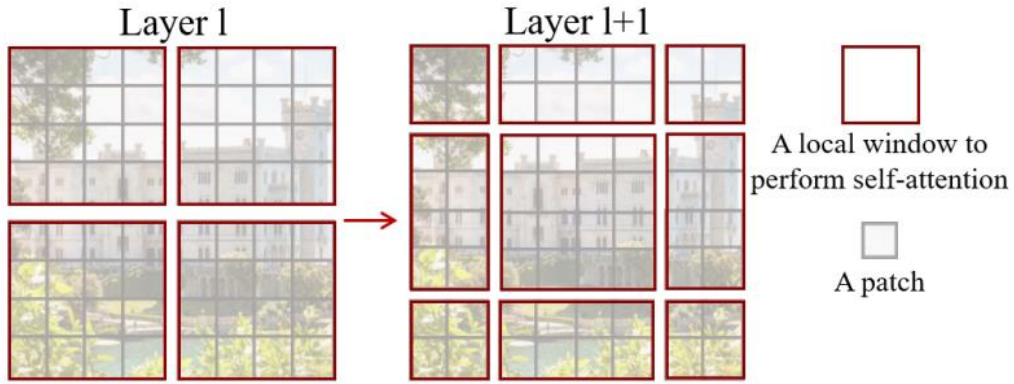
Note: CNN **receptive field** grows only linearly per layer.

SWIN Vision Transformer

• "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows" (Liu, Microsoft, August 2021)

• SWIN builds on ViT by constructing a hierarchical feature representation by starting with small-sized patches and gradually merging them with neighbors in deeper layers. Because the number of patches is fixed, the computational **complexity becomes linear wrt image size**.

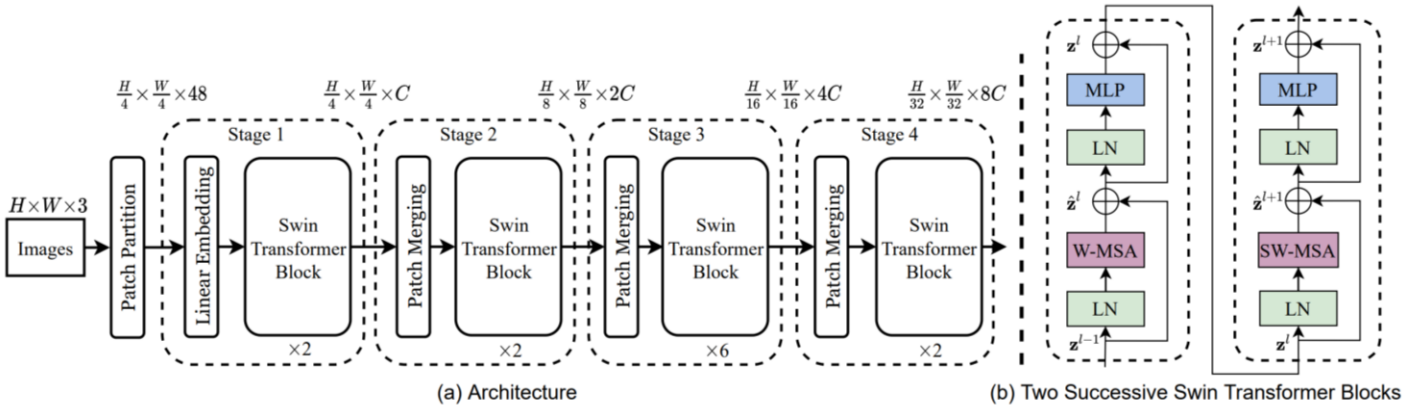




https://arxiv.org/pdf/2103.14030.pdf

SWIN Vision Transformer

(a) Regular ImageNet-1K trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [48]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224^2	84M	16.0G	334.7	82.9
EffNet-B3 [58]	300^2	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380^2	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456^2	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528^2	43M	19.0G	96.9	84.0
EffNet-B7 [58]	600^2	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384^2	307M	190.7G	27.3	76.5
DeiT-S [63]	224^2	22M	4.6G	940.4	79.8
DeiT-B [63]	224^2	86M	17.5G	292.3	81.8
DeiT-B [63]	384^2	86M	55.4G	85.9	83.1
Swin-T	224^2	29M	4.5G	755.2	81.3
Swin-S	224^2	50M	8.7G	436.9	83.0
Swin-B	224^2	88M	15.4G	278.1	83.5
Swin-B	384^2	88M	47.0G	84.7	84.5
(b) ImageNet-22K pre-trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3



(a) Architecture

(b) Two Successive Swin Transformer Blocks

(a) Various frameworks							
Method	Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	#param.	FLOPs	FPS
Cascade Mask R-CNN	R-50	46.3	64.3	50.5	82M	739G	18.0
	Swin-T	**50.5**	**69.3**	**54.9**	86M	745G	15.3
ATSS	R-50	43.5	61.9	47.0	32M	205G	28.3
	Swin-T	**47.2**	**66.5**	**51.3**	36M	215G	22.3
RepPointsV2	R-50	46.5	64.6	50.3	42M	274G	13.6
	Swin-T	**50.0**	**68.5**	**54.2**	45M	283G	12.0
Sparse R-CNN	R-50	44.5	63.4	48.2	106M	166G	21.0
	Swin-T	**47.9**	**67.3**	**52.3**	110M	172G	18.4

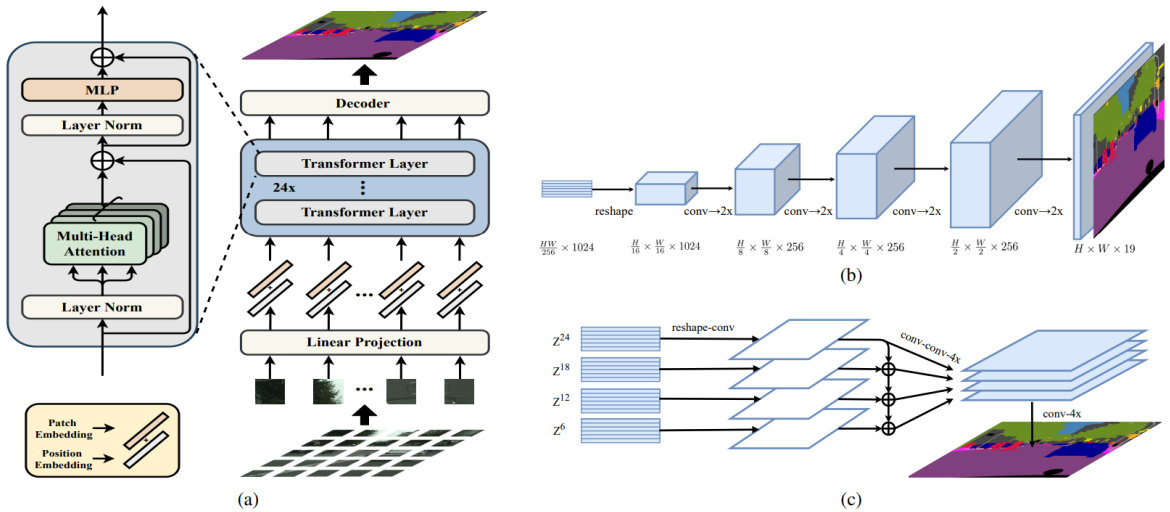
(b) Various backbones w. Cascade Mask R-CNN										
	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}	param	FLOPs	FPS
DeiT-S†	48.0	67.2	51.7	41.4	64.2	44.3	80M	889G	10.4
R50	46.3	64.3	50.5	40.1	61.7	43.4	82M	739G	18.0
Swin-T	**50.5**	**69.3**	**54.9**	**43.7**	**66.6**	**47.1**	86M	745G	15.3
X101-32	48.1	66.5	52.4	41.6	63.9	45.2	101M	819G	12.8
Swin-S	**51.8**	**70.4**	**56.3**	**44.7**	**67.9**	**48.5**	107M	838G	12.0
X101-64	48.3	66.4	52.3	41.7	64.0	45.1	140M	972G	10.4
Swin-B	**51.9**	**70.9**	**56.5**	**45.0**	**68.4**	**48.7**	145M	982G	11.6

ADE20K Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-		
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large‡	50.3	61.7	308M	-	-
UperNet	DeiT-S†	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B‡	51.6	-	121M	1841G	8.7
UperNet	Swin-L‡	**53.5**	**62.8**	234M	3230G	6.2

Table 3. Results of semantic segmentation on the ADE20K val and test set. † indicates additional deconvolution layers are used to produce hierarchical feature maps. ‡ indicates that the model is

SETR ViT

• "Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers" (Zheng, FAIR, 2021) introduce a pure Transformer for semantic segmentation.



Model	T-layers	Hidden size	Att head
T-Base	12	768	12
T-Large	24	1024	16

Table 1. Configuration of Transformer backbone variants.

Method	Pre	Backbone	#Params	40k	80k
FCN [39]	1K	R-101	68.59M	73.93	75.52
Semantic FPN [39]	1K	R-101	47.51M	-	75.80
Hybrid-Base	R	T-Base	112.59M	74.48	77.36
Hybrid-Base	21K	T-Base	112.59M	76.76	76.57
Hybrid-DeiT	21K	T-Base	112.59M	77.42	78.28
SETR-*Naïve*	21K	T-Large	305.67M	77.37	77.90
SETR-*MLA*	21K	T-Large	310.57M	76.65	77.24
SETR-*PUP*	21K	T-Large	318.31M	78.39	79.34
SETR-*PUP*	R	T-Large	318.31M	42.27	-
SETR-*Naïve-Base*	21K	T-Base	87.69M	75.54	76.25
SETR-*MLA-Base*	21K	T-Base	92.59M	75.60	76.87
SETR-*PUP-Base*	21K	T-Base	97.64M	76.71	78.02
SETR-*Naïve-DeiT*	1K	T-Base	87.69M	77.85	78.66
SETR-*MLA-DeiT*	1K	T-Base	92.59M	78.04	78.98
SETR-*PUP-DeiT*	1K	T-Base	97.64M	**78.79**	**79.45**

Table 2. **Comparing SETR variants** on different pre-training strategies and backbones. All experiments are trained on Cityscapes train fine set with batch size 8, and evaluated using the single scale test protocol on the Cityscapes validation set in mean IoU (%) rate. "Pre" denotes the pre-training of transformer part. "R" means the transformer part is randomly initialized.

• Key innovations: Authors introduce **(3) Decoder architectures** (e.g. PUP or "progressive up-sampling") to accommodate dense segmentation output layer.

• Demonstrate effectiveness of using pretrained Transformer (in this case ViT) for downstream task, e.g., dense segmentation, analogous to pre-text training for NLP Transformers.

*https://arxiv.org/pdf/2012.15840.pdf

SAM Optimization

- "Sharpness-Aware Minimization for Efficiently Improving Generalization" (Foret, ICLR 2021)

- Introduces a new paradigm for gradient-based optimizers (SAM):

$$\min_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) + \lambda ||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}),$$

Where the objective (above) is to **minimize a loss function uniformly within an ε-hypersphere** (radius governed by hyperparameter ρ). This objective is solved with a first-order approximation, which, notably requires two backpropgation passes per update.

(*) Importantly, the authors show empirically that **descending in the direction of uniformly minimal regions yields improved generalization**.

Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^{n}\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, Batch size b, Step size $\eta > 0$, Neighborhood size $\rho > 0$.
Output: Model trained with SAM
Initialize weights $\boldsymbol{w}_0$, $t = 0$;
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;
  Compute gradient $\nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$ of the batch's training loss;
  Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ per equation 2;
  Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$;
  Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;
  $t = t + 1$;
end
return $\boldsymbol{w}_t$

Algorithm 1: SAM algorithm

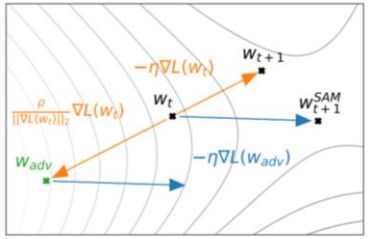


Figure 2: Schematic of the SAM parameter update.

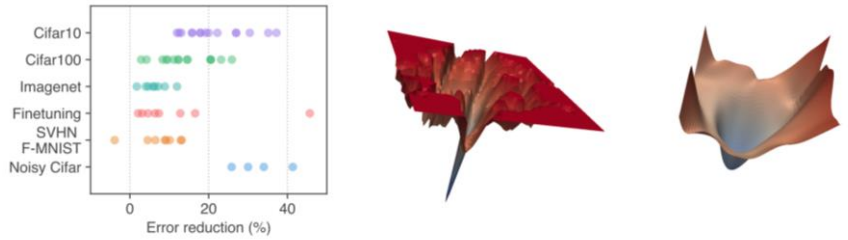
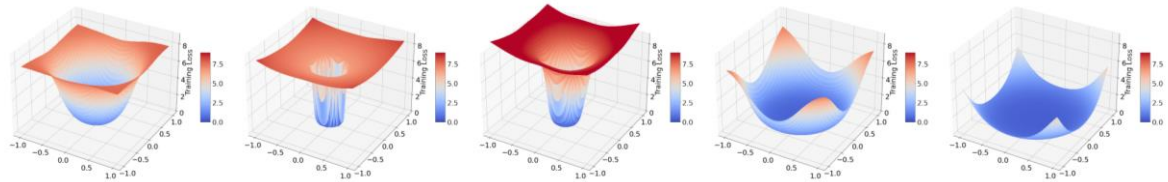


Figure 1: (left) Error rate reduction obtained by switching to SAM. Each point is a different dataset / model / data augmentation. (middle) A sharp minimum to which a ResNet trained with SGD converged. (right) A wide minimum to which the same ResNet trained with SAM converged.

SAM Optimization for ViTs

• More recently, "When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations" (Google, June 2021) demonstrated that **leveraging SAM optimization can obviate the need for large dataset pretraining for ViTs**.



(a) ResNet (b) ViT (c) Mixer (d) ViT-SAM (e) Mixer-SAM

Figure 1: Cross-entropy loss landscapes of ResNet-152, ViT-B/16, and Mixer-B/16. ViT and MLP-Mixer converge to sharper regions than ResNet when trained on ImageNet with the basic Inception-style preprocessing. SAM, a sharpness-aware optimizer, significantly smooths the landscapes.

Table 1: Number of parameters, NTK condition number κ, Hessian dominate eigenvalue λ_{max}, accuracy on ImageNet, and accuracy/robustness on ImageNet-C. ViT and MLP-Mixer suffer divergent κ and converge to sharp regions of big λ_{max}; SAM rescues that and leads to better generalization.

	ResNet-50	ResNet-152	ViT-B/16	ViT-B/16-SAM	Mixer-B/16	Mixer-B/16-SAM
#Params	25M	60M	87M		59M	
NTK κ	2801.6	2801.6	4205.3		14468.0	
Hessian λ_{max}	122.9	179.8	738.8	**20.9**	1644.4	**22.5**
ImageNet (%)	76.0	78.5	74.6	**79.9**	66.4	**77.4**
ImageNet-C (%)	44.6	50.0	46.6	**56.5**	33.8	**48.8**

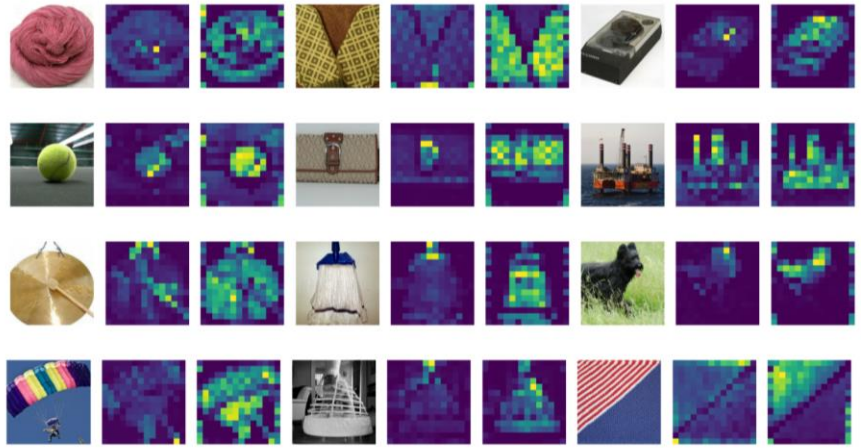


Figure 3: Raw images (**Left**) and attention maps of ViT-S/16 with (**Right**) and without (**Middle**) sharpness-aware optimization. ViT-S/16 with less sharp local optimum contains perceptive segmentation information in its attention maps.

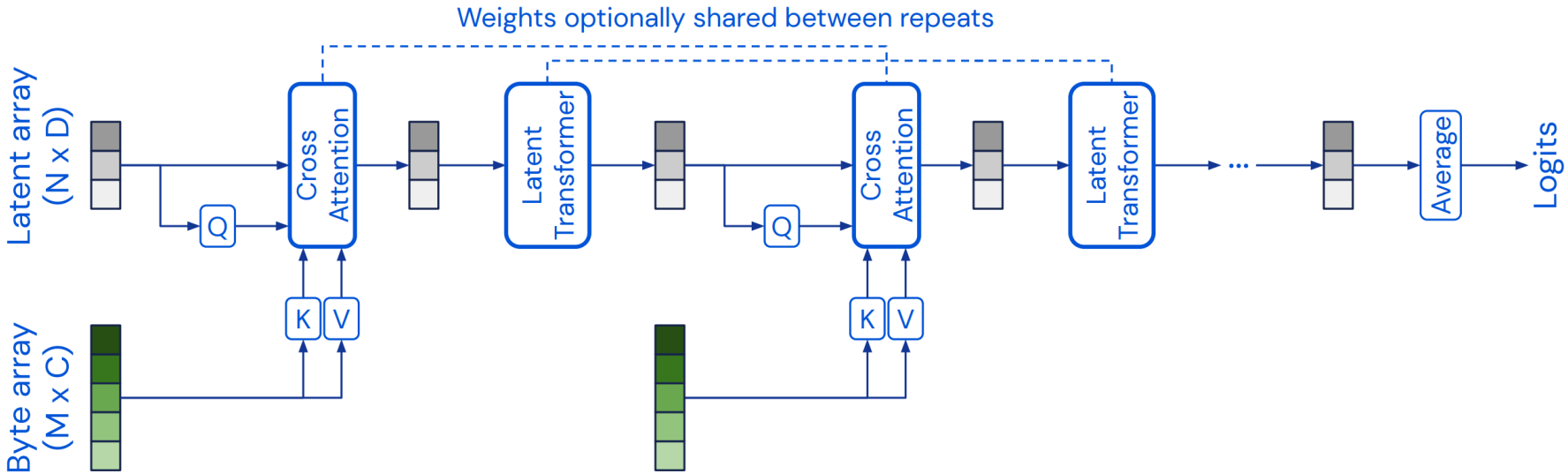
*https://arxiv.org/pdf/2106.01548.pdf

Perceiver

• DeepMind introduced "Perceiver: General Perception with Iterative Attention" (Jaegle, 2021): a model building upon Transformers using **asymmetrical attention** to iteratively distill inputs into a small latent bottleneck (e.g., $224 \times 224 \rightarrow 512$ latent dimensions).

• Perceiver can **handle high dimensional input** and **different modalities** using a single Transformer-based architecture. No need to fine-tune architecture for different modalities, or decide on bespoke fusion strategies, etc. Relies on scalable *Fourier features* for position encoding.

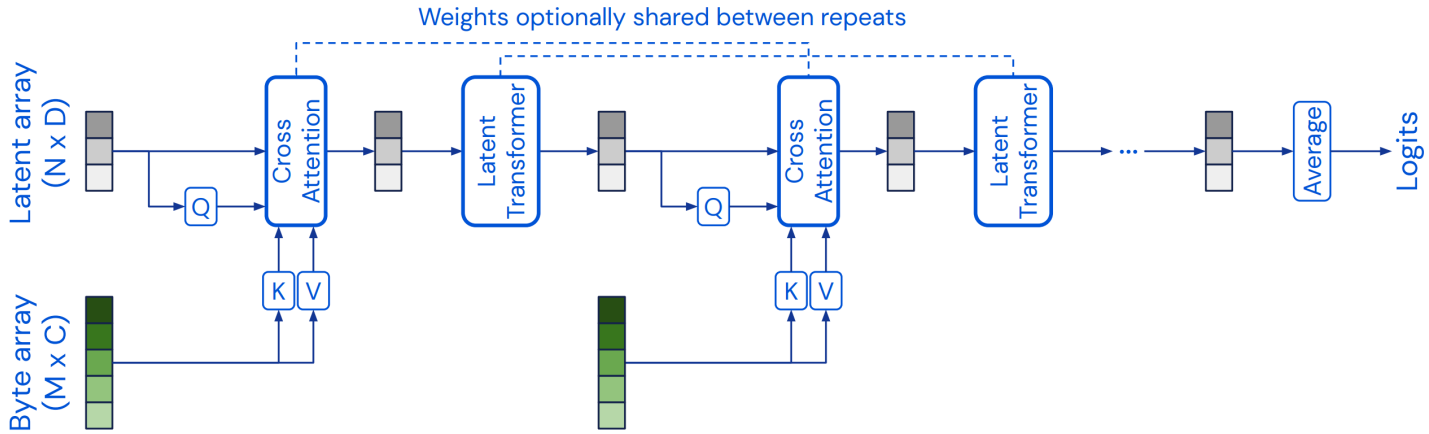
• Model is performant across image, audio, point cloud, video and mixed modality use-cases.



https://arxiv.org/pdf/2103.03206.pdf

Intel Labs: Anthony Rhodes

Perceiver



ResNet-50 (He et al., 2016)	77.6
ViT-B-16 (Dosovitskiy et al., 2021)	77.9
ResNet-50 (FF)	73.5
ViT-B-16 (FF)	76.7
Transformer (64x64, FF)	57.0
Perceiver (FF)	78.0

Table 1. Top-1 validation accuracy (in %) on ImageNet.

	Raw	Perm.	Input RF
ResNet-50 (FF)	73.5	39.4	49
ViT-B-16 (FF)	76.7	61.7	256
Transformer (64x64) (FF)	57.0	57.0	4,096
Perceiver:			
(FF)	78.0	78.0	50,176
(Learned pos.)	70.9	70.9	50,176

Model / Inputs	Audio	Video	A+V
Benchmark (Gemmeke et al., 2017)	31.4	-	-
Attention (Kong et al., 2018)	32.7	-	-
Multi-level Attention (Yu et al., 2018)	36.0	-	-
ResNet-50 (Ford et al., 2019)	38.0	-	-
CNN-14 (Kong et al., 2020)	43.1	-	-
CNN-14 (no balancing & no mixup) (Kong et al., 2020)	37.5	-	-
G-blend (Wang et al., 2020c)	32.4	18.8	41.8
Attention AV-fusion (Fayek & Kumar, 2020)	38.4	25.7	46.2
Perceiver (raw audio)	38.3	25.8	43.5
Perceiver (mel spectrogram)	38.4	25.8	43.2
Perceiver (mel spectrogram - tuned)	-	-	44.2

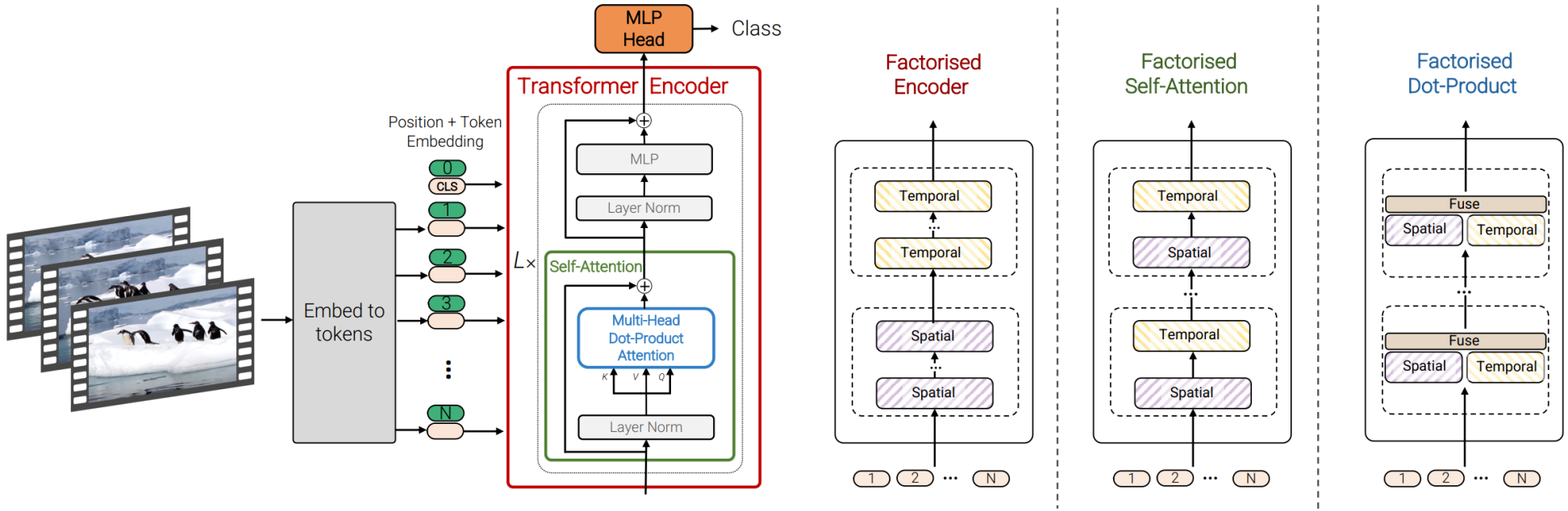
An Overview of Transformers

1. Transformers
2. Vision Transformers (ViTs)
3. Video Vision Transformers

"ViViT: A Video Vision Transformer" (Google, 2021)

- Pure Transformer based model for video classification. Authors propose several variants which factorize the spatial and temporal dimensions of the input. They also show effects of regularization and leveraging pretrained ViTs.

- Key innovations: efficient calculation of attention by factorizing the spatial and temporal dimensions of the input video.



*https://arxiv.org/pdf/2103.15691.pdf

"ViViT: A Video Vision Transformer" (Google, 2021)

- Two ways to embed video clips for attention calculation: (1) **Uniform frame sampling** and (2) **Tubelet embedding**.

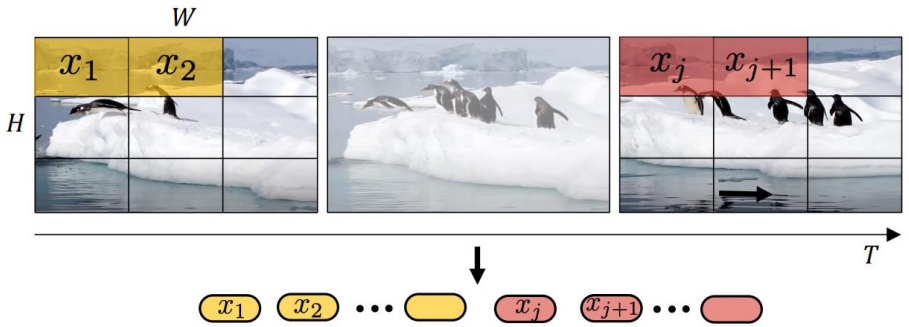


Figure 2: Uniform frame sampling: We simply sample n_t frames, and embed each 2D frame independently following ViT [15].

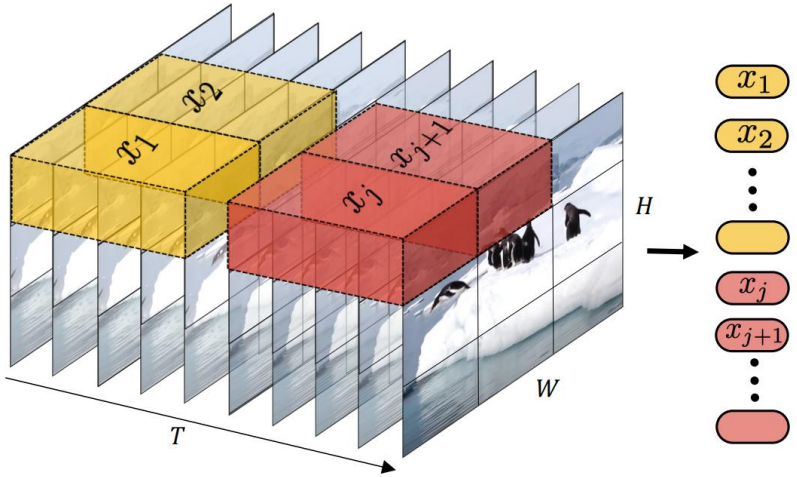


Figure 3: Tubelet embedding. We extract and linearly embed non-overlapping tubelets that span the spatio-temporal input volume.

Table 1: Comparison of input encoding methods using ViViT-B and spatio-temporal attention on Kinetics. Further details in text.

	Top-1 accuracy
Uniform frame sampling	78.5
Tubelet embedding	
Random initialisation [22]	73.2
Filter inflation [6]	77.6
Central frame	79.2

"ViViT: A Video Vision Transformer" (Google, 2021)

- Attention Model 1: **Spatio-Temporal Attention**: Simply model spatio-temporal attention jointly (most computationally intensive).
- Attention Model 2: **Factorized Encoder**: Two separate Transformer encoders: spatial and temporal. Frame-level representations are concatenated together and forwarded through a temporal encoder.

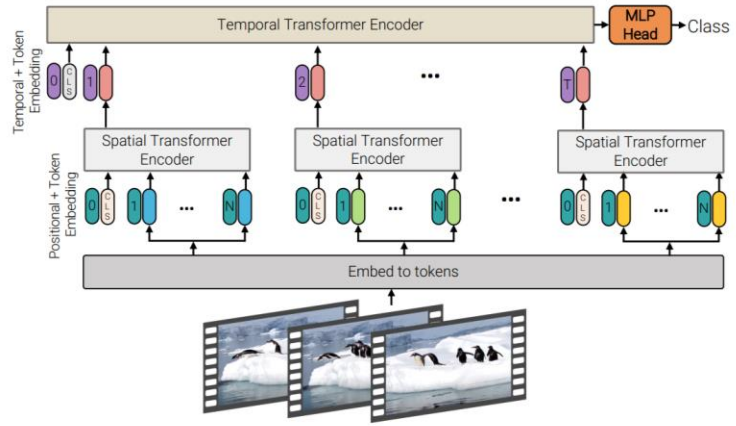


Figure 4: Factorised encoder (Model 2). This model consists of two transformer encoders in series: the first models interactions between tokens extracted from the same temporal index to produce a latent representation per time-index. The second transformer models interactions between time steps. It thus corresponds to a "late fusion" of spatial- and temporal information.

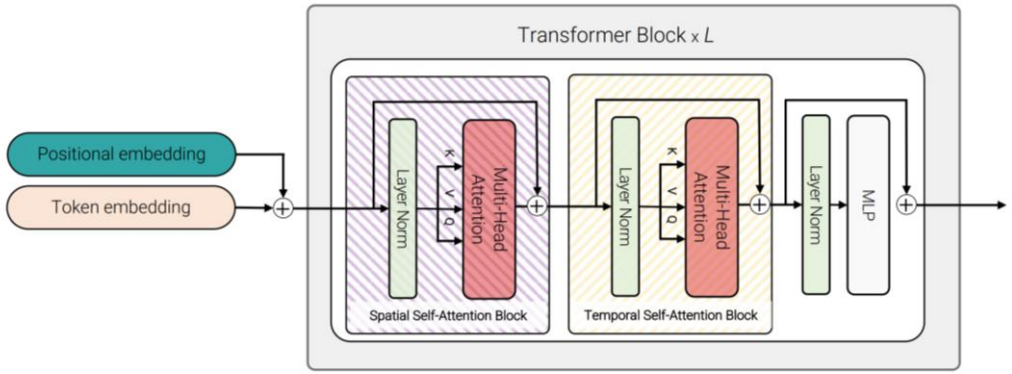


Figure 5: Factorised self-attention (Model 3). Within each transformer block, the multi-headed self-attention operation is factorised into two operations (indicated by striped boxes) that first only compute self-attention spatially, and then temporally.

- Attention Model 3: **Factorized self-attention**: like Model 1, but more efficient. Factorize attention so that the first layer only compute SA spatially (same temporal index); next layer only computes attention temporally (among all tokens extracted from same spatial index).

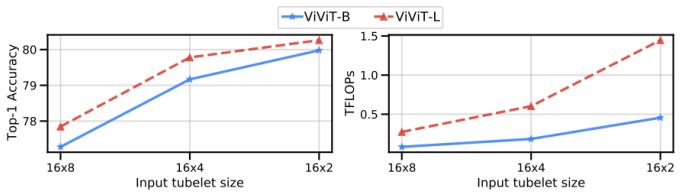
"ViViT: A Video Vision Transformer" (Google, 2021)

Table 2: Comparison of model architectures using ViViT-B as the backbone, and tubelet size of 16×2. We report Top-1 accuracy on Kinetics 400 (K400) and action accuracy on Epic Kitchens (EK). Runtime is during inference on a TPU-v3.

	K400	EK	FLOPs ($\times 10^9$)	Params ($\times 10^6$)	Runtime (ms)
Model 1: Spatio-temporal	80.0	43.1	455.2	88.9	58.9
Model 2: Fact. encoder	78.8	43.7	284.4	100.7	17.4
Model 3: Fact. self-attention	77.4	39.1	372.3	117.3	31.7

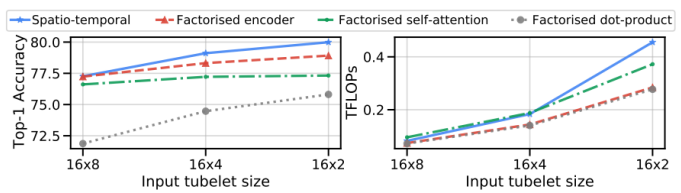
Table 4: The effect of progressively adding regularisation (each row includes all methods above it) on Top-1 action accuracy on Epic Kitchens. We use a Factorised encoder model with tubelet size 16×2.

	Top-1 accuracy
Random crop, flip, colour jitter	38.4
+ Kinetics 400 initialisation	39.6
+ Stochastic depth [28]	40.2
+ Random augment [10]	41.1
+ Label smoothing [58]	43.1
+ Mixup [79]	43.7



(a) Accuracy (b) Compute

Figure 7: The effect of the backbone architecture on (a) accuracy and (b) computation on Kinetics 400, for the spatio-temporal attention model (Model 1).



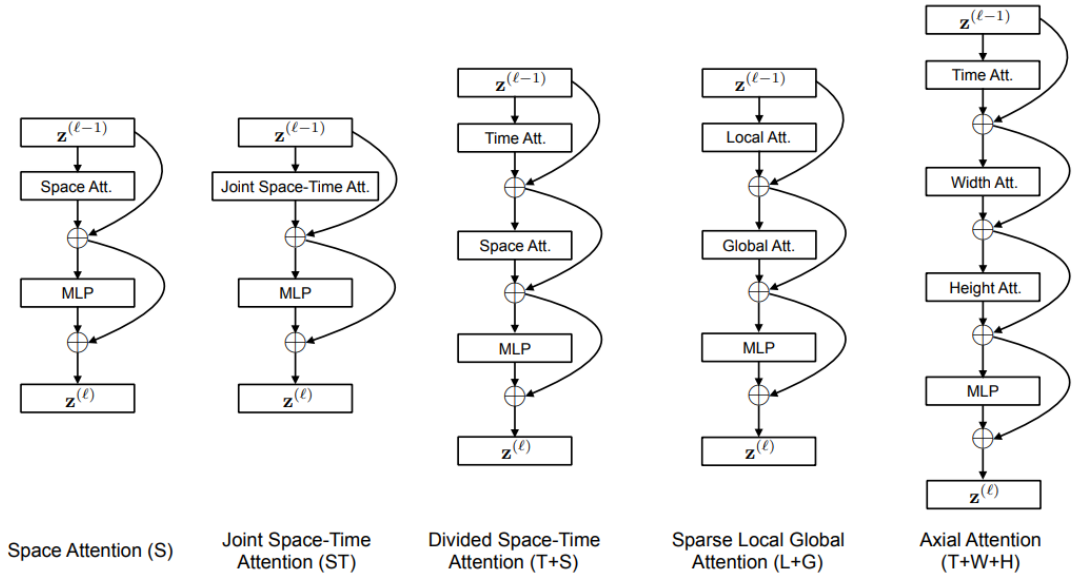
(a) Accuracy (b) Compute

Figure 8: The effect of varying the number of temporal tokens on (a) accuracy and (b) computation on Kinetics 400, for different variants of our model with a ViViT-B backbone.

"Is Space-Time Attention All you Need for Video Understanding?" (Facebook AI, 2021)

- Authors also introduce a pure Transformer-based video classifier. Key observation: self-attention requires computing similarity for all pairs of tokens; consequently, efficient calculation of attention is vital.

- They propose several schema:



Attention	Params	K400	SSv2
Space	85.9M	76.9	36.6
Joint Space-Time	85.9M	77.4	58.5
Divided Space-Time	121.4M	**78.0**	**59.5**
Sparse Local Global	121.4M	75.9	56.3
Axial	156.8M	73.5	56.2

Model	Pretrain	K400 Training Time (hours)	K400 Acc.	Inference TFLOPs	Params
I3D 8x8 R50	ImageNet-1K	444	71.0	1.11	28.0M
I3D 8x8 R50	ImageNet-1K	1440	73.4	1.11	28.0M
SlowFast R50	ImageNet-1K	448	70.0	1.97	34.6M
SlowFast R50	ImageNet-1K	3840	75.6	1.97	34.6M
SlowFast R50	N/A	6336	76.4	1.97	34.6M
TimeSformer	ImageNet-1K	**416**	75.8	**0.59**	121.4M
TimeSformer	ImageNet-21K	**416**	**78.0**	**0.59**	121.4M

- (ST): Jointly calculate space-time attention (most computationally intensive).
- (T+S): Separate spatial and temporal attention sequentially.
- (L+G): First compute local attention by considering neighboring patches, then global attention over entire clip using stride = 2 patches.
- (T+W+H): Decomposes attention over 3 dimensions: time, width and height.

https://arxiv.org/pdf/2102.05095.pdf

Intel Labs: Anthony Rhodes

Landscape Analysis of Vision Transformers

(+) Lack of inductive bias, good fit for data-drive applications, SOTA results, many topical innovations, can model long-term statistical dependencies, computational efficiency, multi-modal fusion potential, faster to train.

Some essential considerations:
(*) Large models, require large datasets
(*) Innovation potential with efficient calculation/use of attention, multi-modal data fusion
(*) Pre-text training, downstream fine-tuning
(*) Fit for weakly-supervised data?
(*) Continuous learning potential?
(*) Scaling and generalizability?