MLP-Mixer & gMLP

Kurtsev Dmitriy

CMC MSU

April 2022

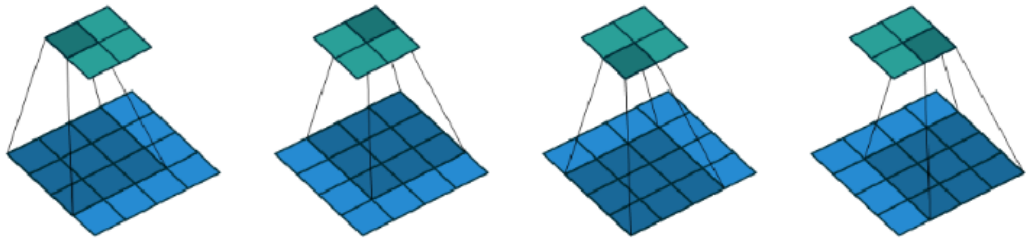
Content

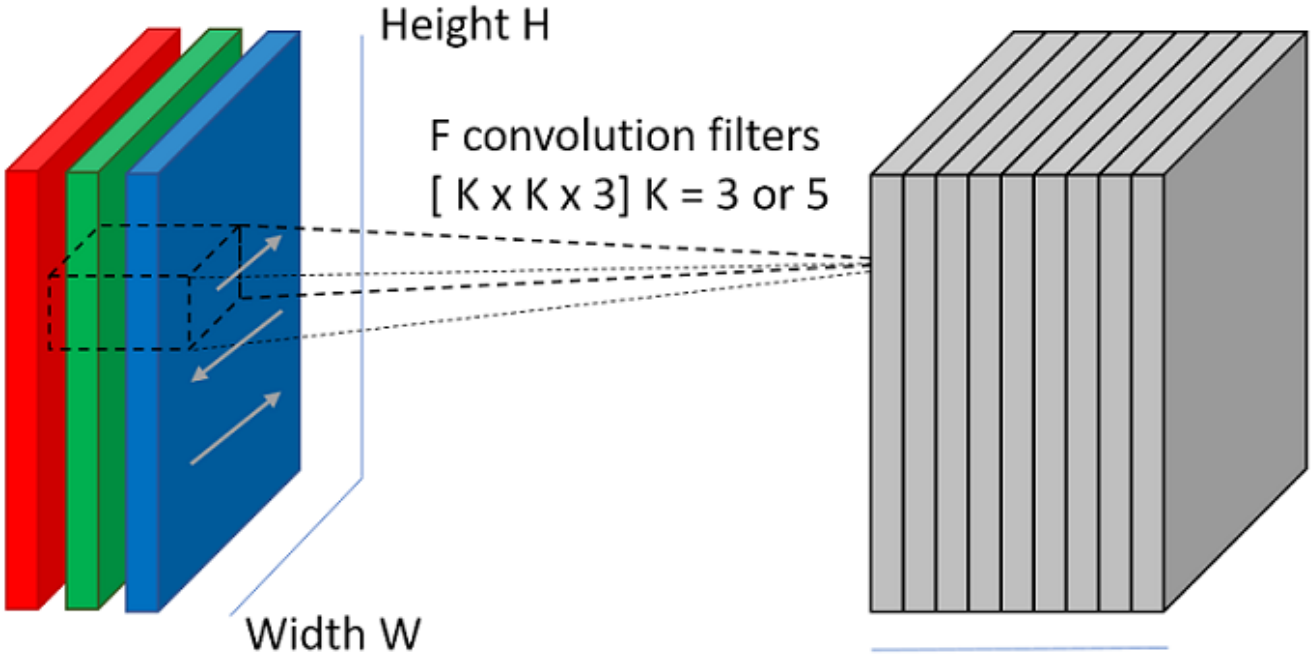
1. CV
   - CNN
   - ViT
   - MLP
2. NLP
   - Transformers
   - MLP

Convolutions

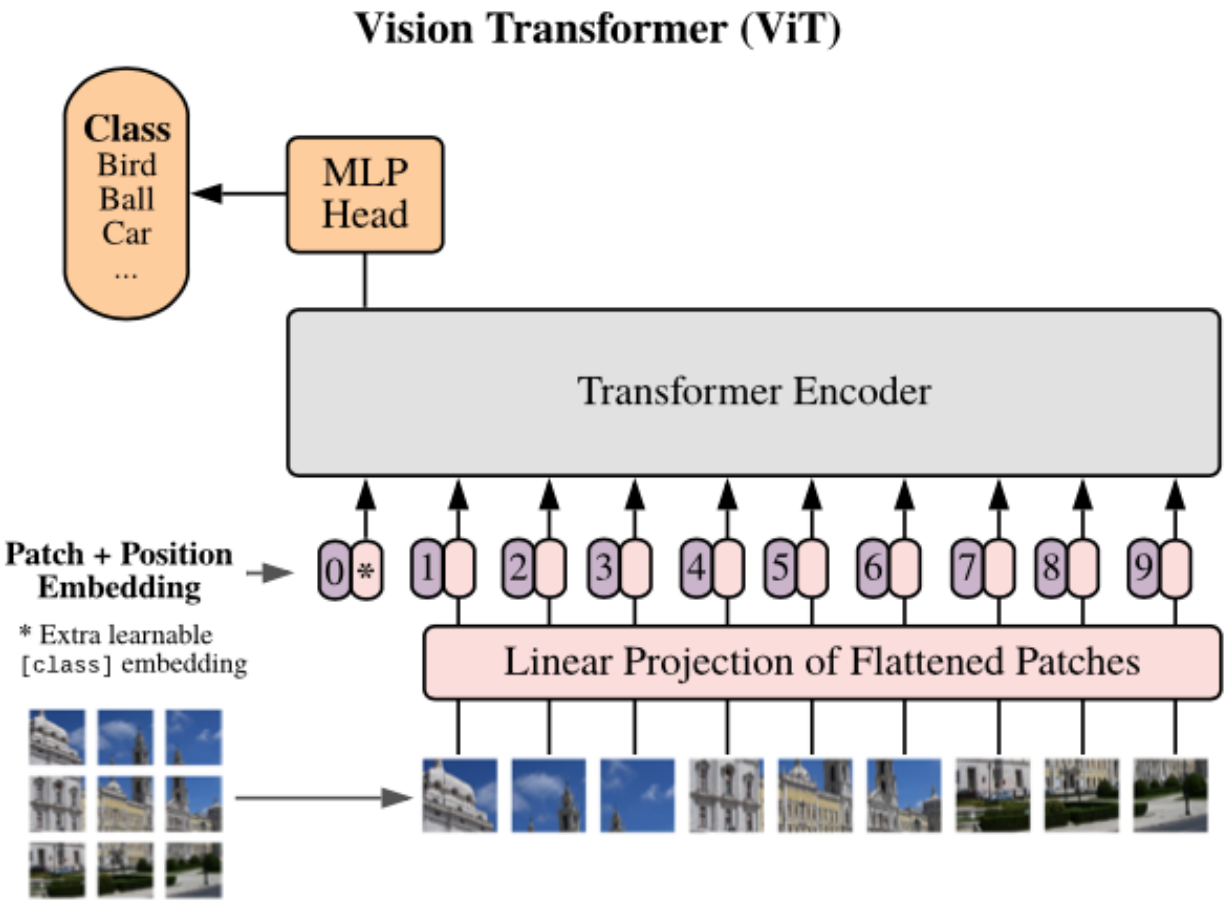
Convolutions in the discrete case:

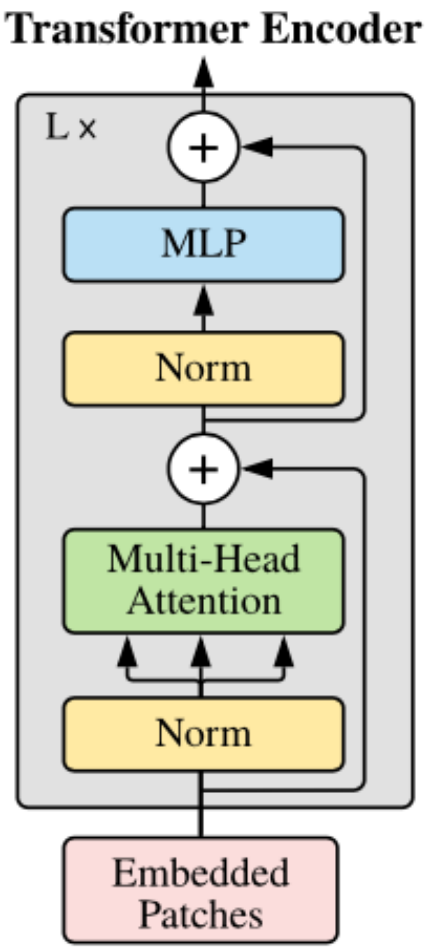
$$(f * g)(i) = \sum_{j \in \mathbb{Z}} f(j) g(i - j)$$





Vision Transformer





MLP-Mixer

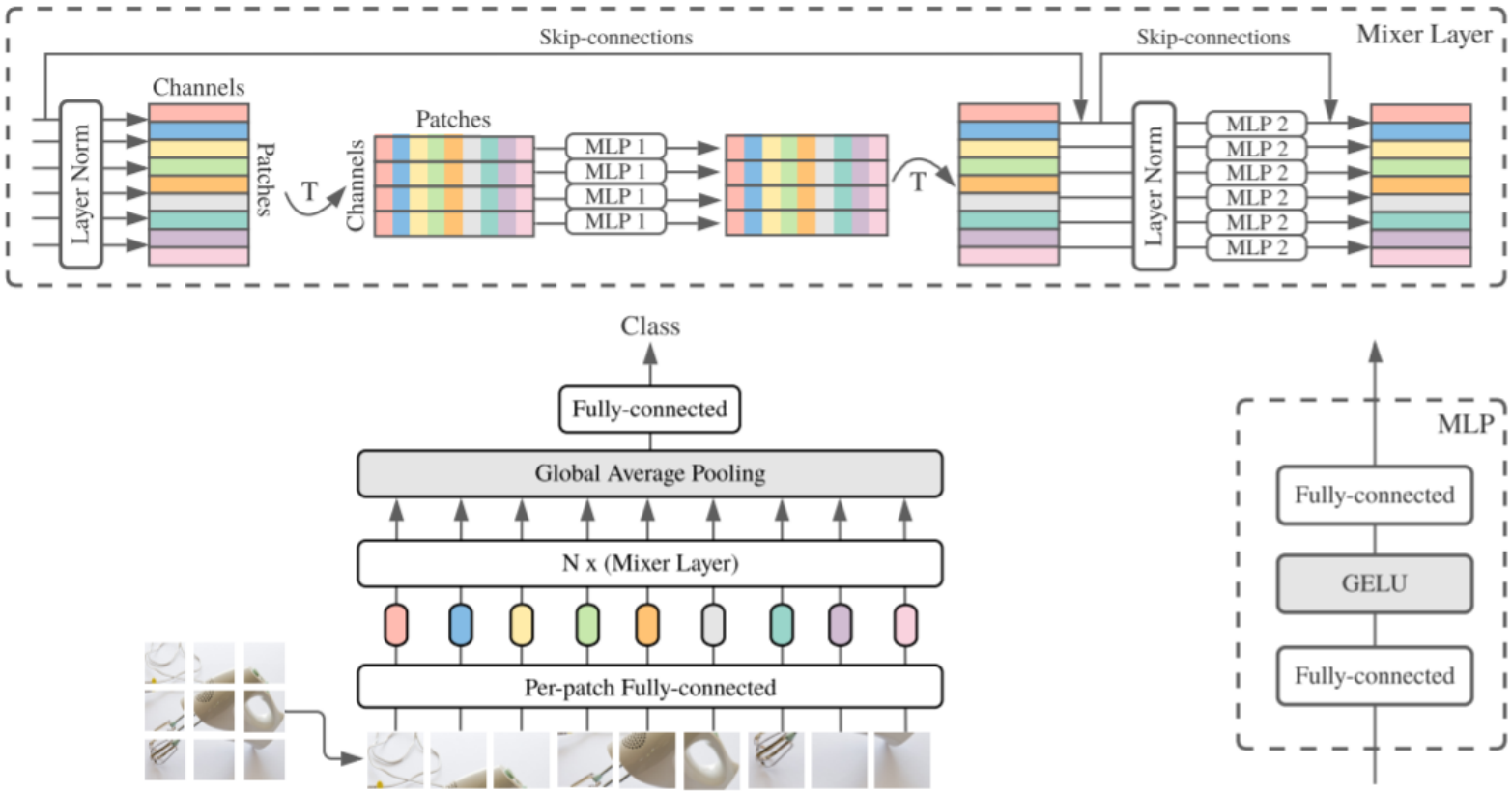


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer norm on the channels.

Mixer architecture

Input image with resolution (H, W), Patch resolution is $(P, P) \Rightarrow S = \frac{HW}{P^2}$
Input matrix $\mathbf{X} \in \mathbb{R}^{S \times C}$

Each block is defined as:
$U_{*,i} = X_{*,i} + W_2\ \sigma(W_1\ LayerNorm(X)_{*,i}), \quad i = \overline{1, C}$ - token-mixing
$Y_{j,*} = U_{j,*} + W_4\ \sigma(W_3\ LayerNorm(U)_{j,*}), \quad j = \overline{1, S}$ - channel-mixing
where $\sigma = GELU()$, $W_1 \in \mathbb{R}^{D_S \times S}$, $W_2 \in \mathbb{R}^{S \times D_S}$, $W_3 \in \mathbb{R}^{D_C \times C}$, $W_4 \in \mathbb{R}^{C \times D_C}$

$$GELU(x) = x\Phi(x) = 0.5x(1 + tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$

Token-mixing have $2D_S SC$ operations

Similarities and differences

Two types of layers - channel-mixing(i) and token-mixing(ii).
CNN: both (i) and (ii) at once.
Transformers: both (i) and (ii) at once.
MLP-Mixer: do separately

At Mixer no *self-attention* blocks, no *positional embedding*.

Google's results

Specification	S/32	S/16	B/32	B/16	L/32	L/16	H/14
Number of layers	8	8	12	12	24	24	32
Patch resolution $P \times P$	32×32	16×16	32×32	16×16	32×32	16×16	14×14
Hidden size C	512	512	768	768	1024	1024	1280
Sequence length S	49	196	49	196	49	196	256
MLP dimension D_C	2048	2048	3072	3072	4096	4096	5120
MLP dimension D_S	256	256	384	384	512	512	640
Parameters (M)	19	18	60	59	206	207	431

	ImNet top-1	ReaL top-1	Avg 5 top-1	VTAB-1k 19 tasks	Throughput img/sec/core	TPUv3 core-days
Pre-trained on ImageNet-21k (public)						
● HaloNet [51]	85.8	—	—	—	120	0.10k
● Mixer-L/16	84.15	87.86	93.91	74.95	105	0.41k
● ViT-L/16 [14]	85.30	88.62	94.39	72.72	32	0.18k
● BiT-R152x4 [22]	85.39	—	94.04	70.64	26	0.94k
Pre-trained on JFT-300M (proprietary)						
● NFNet-F4+ [7]	89.2	—	—	—	46	1.86k
● Mixer-H/14	87.94	90.18	95.71	75.33	40	1.01k
● BiT-R152x4 [22]	87.54	90.54	95.33	76.29	26	9.90k
● ViT-H/14 [14]	88.55	90.72	95.97	77.63	15	2.30k
Pre-trained on unlabelled or weakly labelled data (proprietary)						
● MPL [34]	90.0	91.12	—	—	—	20.48k
● ALIGN [21]	88.64	—	—	79.99	15	14.82k

My results v.s. Google

	CIFAR-10	CIFAR-100	Pets
Pre-trained on ImageNet-21k			
my MLP-B/16	97.7	89.7	97.4
their MLP-B/16	96.8	-	-
ViT-B/16	98.8	91.9	-
ResNet152	98.9	88.5	96.6
From scratch			
my MLP-B/16	82.3	68.8	79.4
ViT-B/16	98.6	89.1	93.1
ResNet152	98.2	87.8	93.3

The role of the model scale

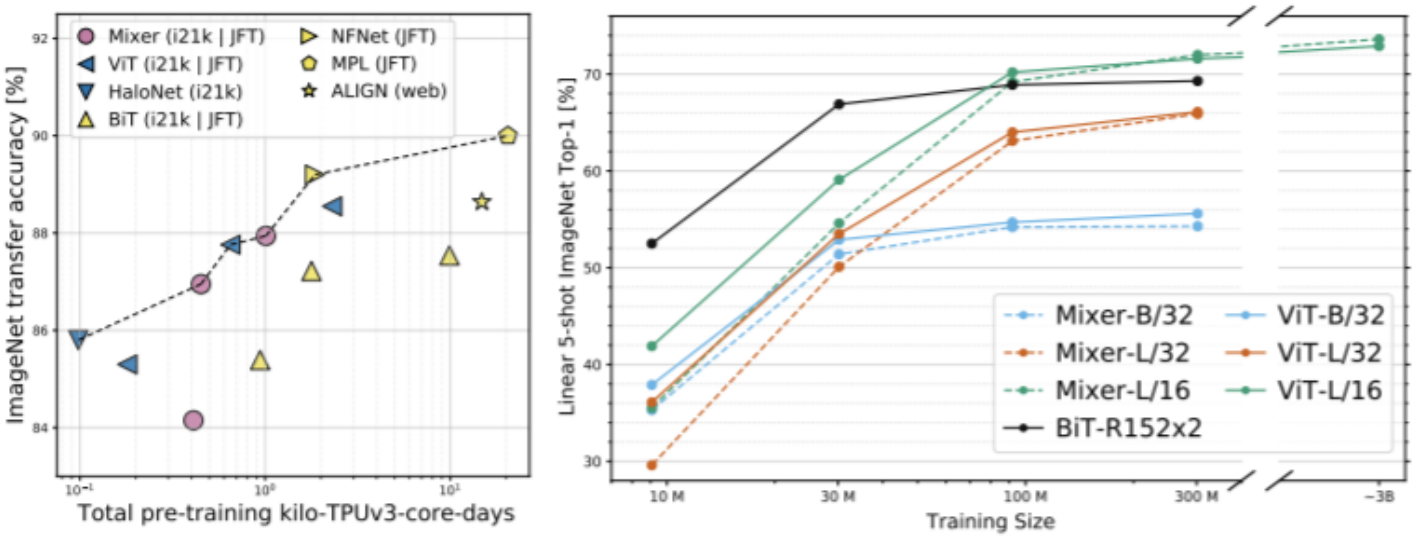


Figure 2: **Left:** ImageNet accuracy/training cost Pareto frontier (dashed line) for the SOTA models in Table 2. Models are pre-trained on ImageNet-21k, or JFT (labelled, or pseudo-labelled for MPL), or web image text pairs. Mixer is as good as these extremely performant ResNets, ViTs, and hybrid models, and sits on frontier with HaloNet, ViT, NFNet, and MPL. **Right:** Mixer (solid) catches or exceeds BiT (dotted) and ViT (dashed) as the data size grows. Every point on a curve uses the same pre-training compute; they correspond to pre-training on 3%, 10%, 30%, and 100% of JFT-300M for 233, 70, 23, and 7 epochs, respectively. Additional points at ∼3B correspond to pre-training on an even larger JFT-3B dataset for the same number of total steps. Mixer improves more rapidly with data than ResNets, or even ViT. The gap between large Mixer and ViT models shrinks.

Content

Self-Attention

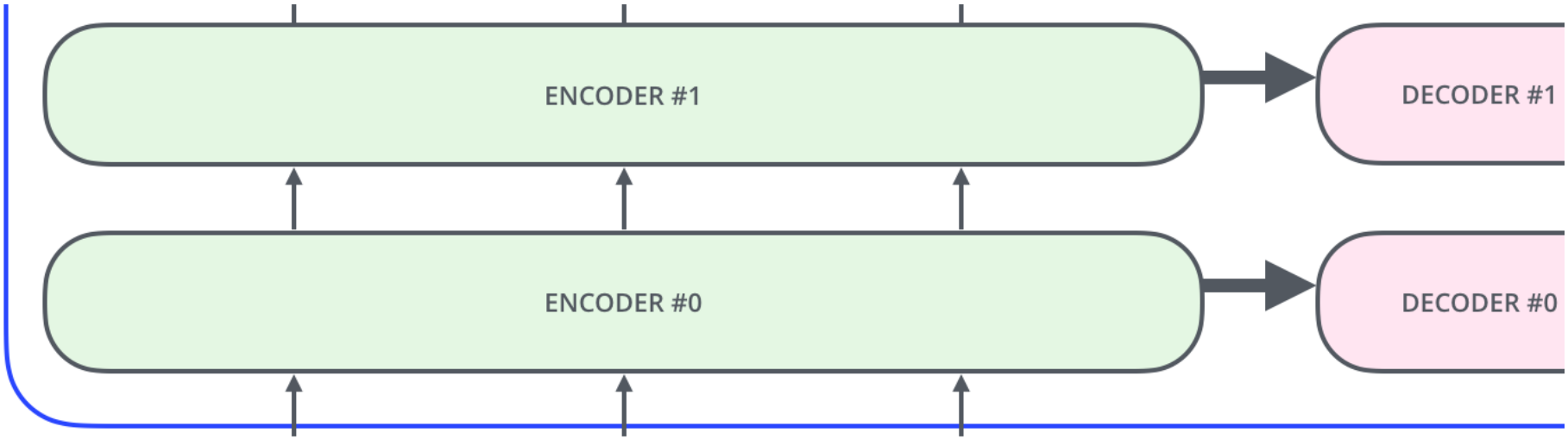
Input matrix $X \in \mathbb{R}^{S \times C}$, S tokens, embedding dimension C.

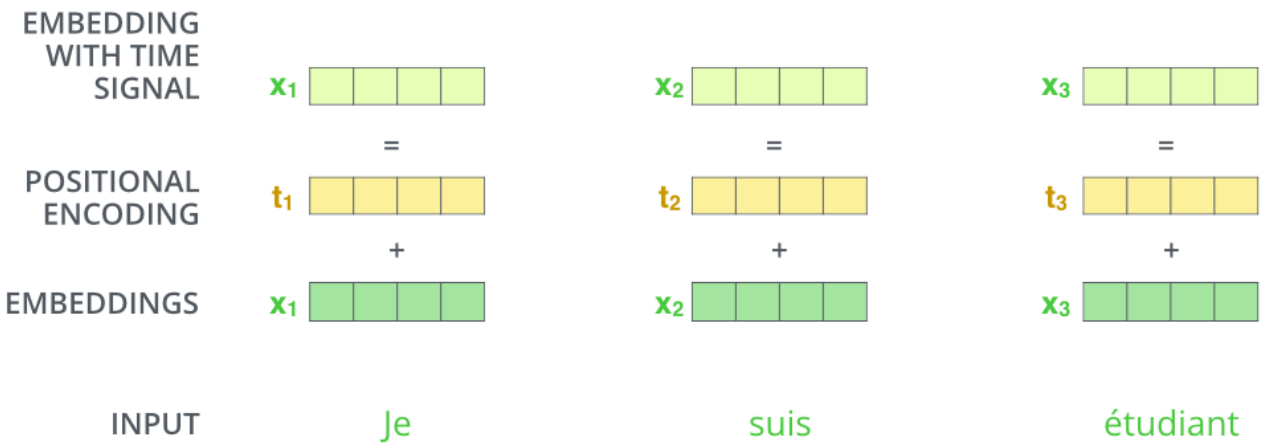
$K_{S \times d} = X_{S \times C}\ W^K_{C \times d} - keys$
$Q_{S \times d} = X_{S \times C}\ W^Q_{C \times d} - query$
$V_{S \times d} = X_{S \times C}\ W^V_{C \times d} - values$
$z = head(X|W^K, W^Q, W^V) = softmax(\frac{QK^T}{\sqrt{d}})\ V$

Multi-Headed Attention
Some different W^K_i, W^Q_i, W^V_i, for $i = \overline{1, l} \Rightarrow$ we have different z_i, for $i = \overline{1, l}$
$z = concat[z_i, axis = 0]^l_{i=1}$
$output = zW$

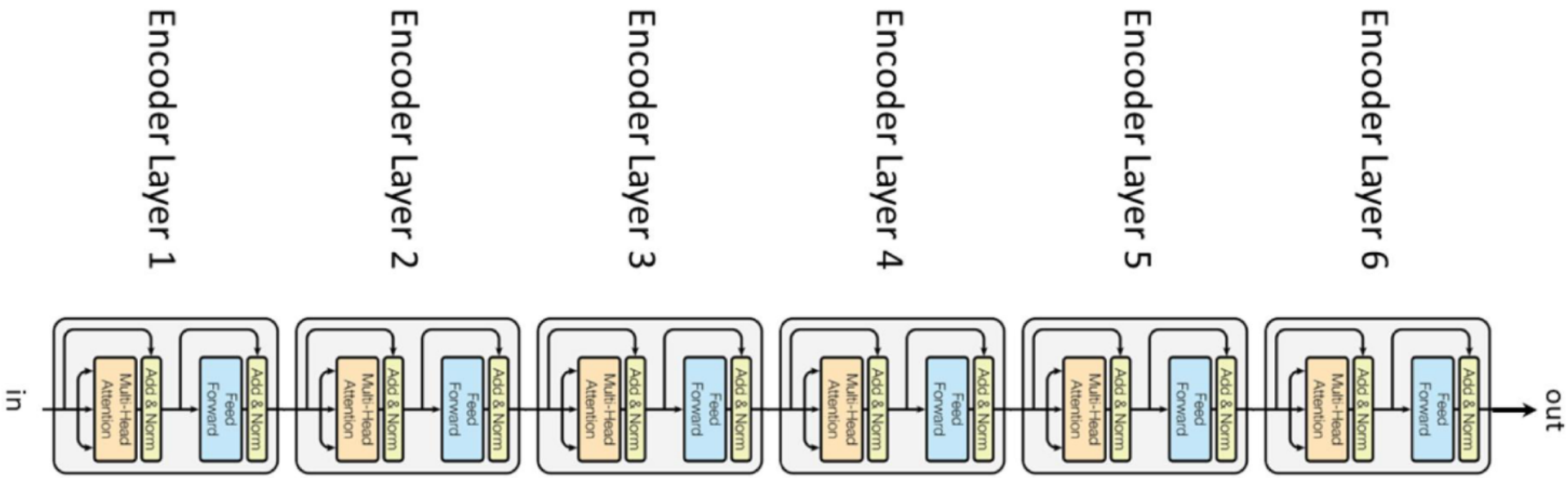
Self-attention have $3SCd + 2S^2d$ operations

Positional Encoding

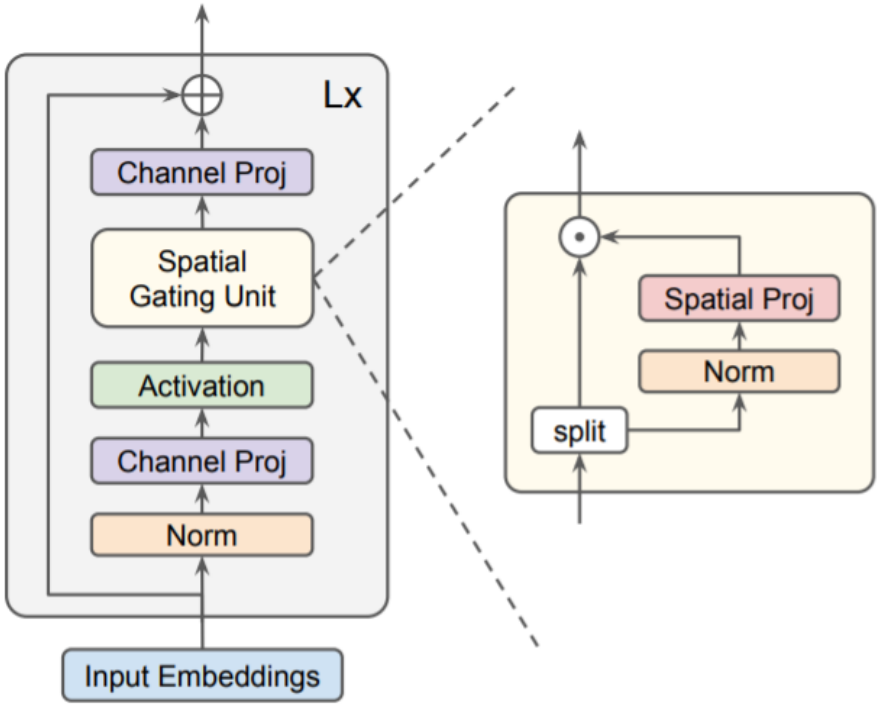




Encoder


Encoder Layer 6
Encoder Layer 5
Encoder Layer 4
Encoder Layer 3
Encoder Layer 2
Encoder Layer 1
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
in
out

gMLP



Pseudo-code for the gMLP block

```
def gmlp_block(x, d_model, d_ffn):
  shortcut = x
  x = norm(x, axis="channel")
  x = proj(x, d_ffn, axis="channel")
  x = gelu(x)
  x = spatial_gating_unit(x)
  x = proj(x, d_model, axis="channel")
  return x + shortcut

def spatial_gating_unit(x):
  u, v = split(x, axis="channel")
  v = norm(v, axis="channel")
  n = get_dim(v, axis="spatial")
  v = proj(v, n, axis="spatial", init_bias=1)
  return u * v
```

Figure 1: Overview of the gMLP architecture with Spatial Gating Unit (SGU). The model consists of a stack of L blocks with identical structure and size. All projection operations are linear and "$\odot$" refers to element-wise multiplication (linear gating). The input and output protocols follow BERT for NLP and ViT for vision. Unlike Transformers, gMLPs do not require positional encodings, nor is it necessary to mask out the paddings during NLP finetuning.

gMLP architecture

Input matrix $X \in \mathbb{R}^{S \times C}$, S tokens, embedding dimension C.

Each block is defined as:
$Z = GELU(XU), \quad \widetilde{Z} = s(Z), \quad Y = \widetilde{Z}V$
$U \in \mathbb{R}^{C \times D_C}, V \in \mathbb{R}^{D_C \times C}$

Spatial Gating Unit:
$f(Z) = WZ + b, \quad W \in \mathbb{R}^{S \times S}, S$ - sequence length
$s(Z) = Z \odot f(Z)$, where $\odot$ - element-wise multiplication.

It's effective to split Z into two independent parts (Z_1, Z_2) along the channel dimension:
$s(Z) = Z_1 \odot f(Z_2)$

SGU have $D_C S^2$ operations

Comparison

Model(Base)	Complexity	Params(M)
MLP-Mixer	$2D_S SC$	59
gMLP in CV	$D_C S^2$	73
gMLP in NLP	$D_C S^2$	102
ViT	$3SCd + 2S^2 d$	86
BERT	$3SCd + 2S^2 d$	110

Experiments in CV

Table 2: ImageNet-1K results without extra data.

Model	ImageNet Top-1 (%)*	Input Resolution	Params (M)	MAdds (B)
	ConvNets			
ResNet-152 [16]	78.3	224	60	11.3
RegNetY-8GF [39]	81.7	224	39	8.0
EfficientNet-B0 [17]	77.1	224	5	0.39
EfficientNet-B3 [17]	81.6	300	12	1.8
EfficientNet-B7 [17]	84.3	600	66	37.0
NFNet-F0 [33]	83.6	192	72	12.4
	Transformers			
ViT-B/16 [7]	77.9	384	86	55.4
ViT-L/16 [7]	76.5	384	307	190.7
DeiT-Ti [8] (ViT+reg)	72.2	224	5	1.3
DeiT-S [8] (ViT+reg)	79.8	224	22	4.6
DeiT-B [8] (ViT+reg)	81.8	224	86	17.5
	MLP-like†			
Mixer-B/16 [20]	76.4	224	59	12.7
Mixer-B/16 (our setup)	77.3	224	59	12.7
Mixer-L/16 [20]	71.8	224	207	44.8
ResMLP-12 [22]	76.6	224	15	3.0
ResMLP-24 [22]	79.4	224	30	6.0
ResMLP-36 [22]	79.7	224	45	8.9
gMLP-Ti (ours)	72.3	224	6	1.4
gMLP-S (ours)	79.6	224	20	4.5
gMLP-B (ours)	81.6	224	73	15.8

* Standard deviation across multiple independent runs is around 0.1.

† Tokenization & embedding process at the stem can be viewed as a convolution.

Masked Language Modeling

Pre-train on C4/RealNews dataset

The input/output protocol for both pre-training and fine-tuning follows BERT.
Do not use positional encodings.

Google's results

Model	Perplexity*	Params (M)
$\text{BERT}_{\text{base}}$	4.37	110
$\text{BERT}_{\text{base}}$ + rel pos	4.26	110
$\text{BERT}_{\text{base}}$ + rel pos - attn	5.64	96
MLP-Mixer	5.34	112
Linear gMLP, $s(Z) = f(Z)$	5.14	92
Additive gMLP, $s(Z) = Z + f(Z)$	4.97	92
Multiplicative gMLP, $s(Z) = Z \odot f(Z)$	4.53	92
Multiplicative, Split gMLP, $s(Z) = Z_1 \odot f(Z_2), Z = Z_1 \| Z_2$	4.35	102

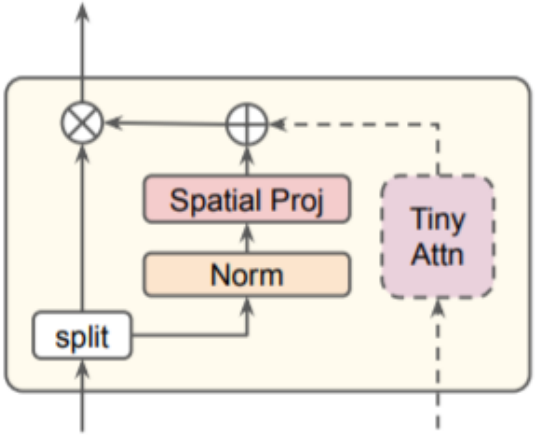
* Standard deviation across multiple independent runs is around 0.01.

Model	#L	Params (M)	Perplexity	SST-2	MNLI-m
Transformer	6+6	67	**4.91**	90.4	81.5
gMLP	18	59	5.25	91.2	77.7
Transformer	12+12	110	**4.26**	91.3	83.3
gMLP	36	102	4.35	92.3	80.9
Transformer	24+24	195	3.83	92.1	85.2
gMLP	72	187	**3.79**	93.5	82.8
Transformer	48+48	365	3.47	92.8	86.3
gMLP	144	357	**3.43**	95.1	84.6

My results v.s. Google

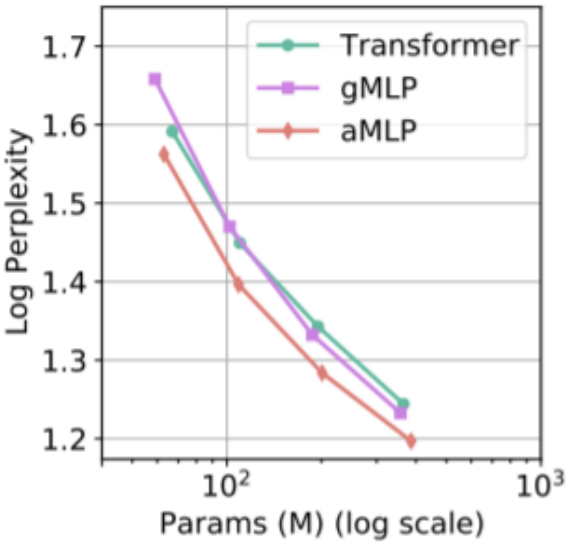
L = 18	SST-2	MNLI-m
my gMLP	80.4	65.1
my gMLP + BERT	88.1	74.6
my gMLP_split + BERT	88.6	75.2
their gMLP_split	91.2	77.7

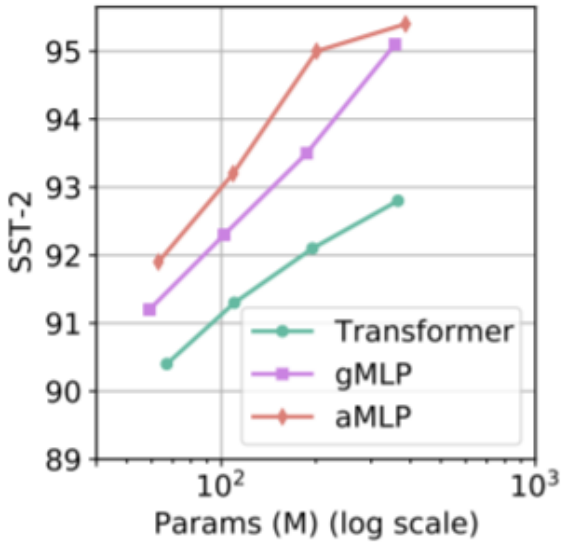
Tiny Attention

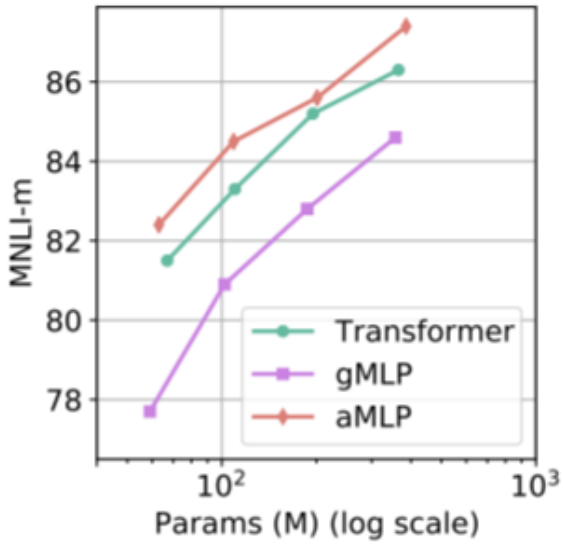


Pseudo-code for the tiny attention module

```
def tiny_attn(x, d_out, d_attn=64):
  qkv = proj(x, 3 * d_attn, axis="channel")
  q, k, v = split(qkv, 3, axis="channel")
  w = einsum("bnd,bmd->bnm", q, k)
  a = softmax(w * rsqrt(d_attn))
  x = einsum("bnm,bmd->bnd", a, v)
  return proj(x, d_out, axis="channel")
```







aMLP v.s. gMLP

Table 6: Pretraining perplexities and dev-set results for finetuning. "ours" indicates models trained using our setup. We report accuracies for SST-2 and MNLI, and F1 scores for SQuAD v1.1/2.0.

	Perplexity	SST-2	MNLI (m/mm)	SQuAD v1.1	SQuAD v2.0	Attn Size	Params (M)
$\text{BERT}_{\text{base}}$ [2]	–	92.7	84.4/-	88.5	76.3	768 (64 × 12)	110
$\text{BERT}_{\text{base}}$ (ours)	4.17	93.8	85.6/85.7	90.2	78.6	768 (64 × 12)	110
$\text{gMLP}_{\text{base}}$	4.28	94.2	83.7/84.1	86.7	70.1	–	130
$\text{aMLP}_{\text{base}}$	3.95	93.4	85.9/85.8	90.7	80.9	64	109
$\text{BERT}_{\text{large}}$ [2]	–	93.7	86.6/-	90.9	81.8	1024 (64 × 16)	336
$\text{BERT}_{\text{large}}$ (ours)	3.35	94.3	87.0/87.4	92.0	81.0	1024 (64 × 16)	336
$\text{gMLP}_{\text{large}}$	3.32	94.8	86.2/86.5	89.5	78.3	–	365
$\text{aMLP}_{\text{large}}$	3.19	94.8	88.4/88.4	92.2	85.4	128	316
$\text{gMLP}_{\text{xlarge}}$	2.89	95.6	87.7/87.7	90.9	82.1	–	941

Literature

https://arxiv.org/pdf/2105.08050.pdf
https://arxiv.org/pdf/2105.01601.pdf