MLBBQ October 8
Alex Fedorov

MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy

https://arxiv.org/abs/2105.01601v4

Architectures

Mixing features in modern architectures

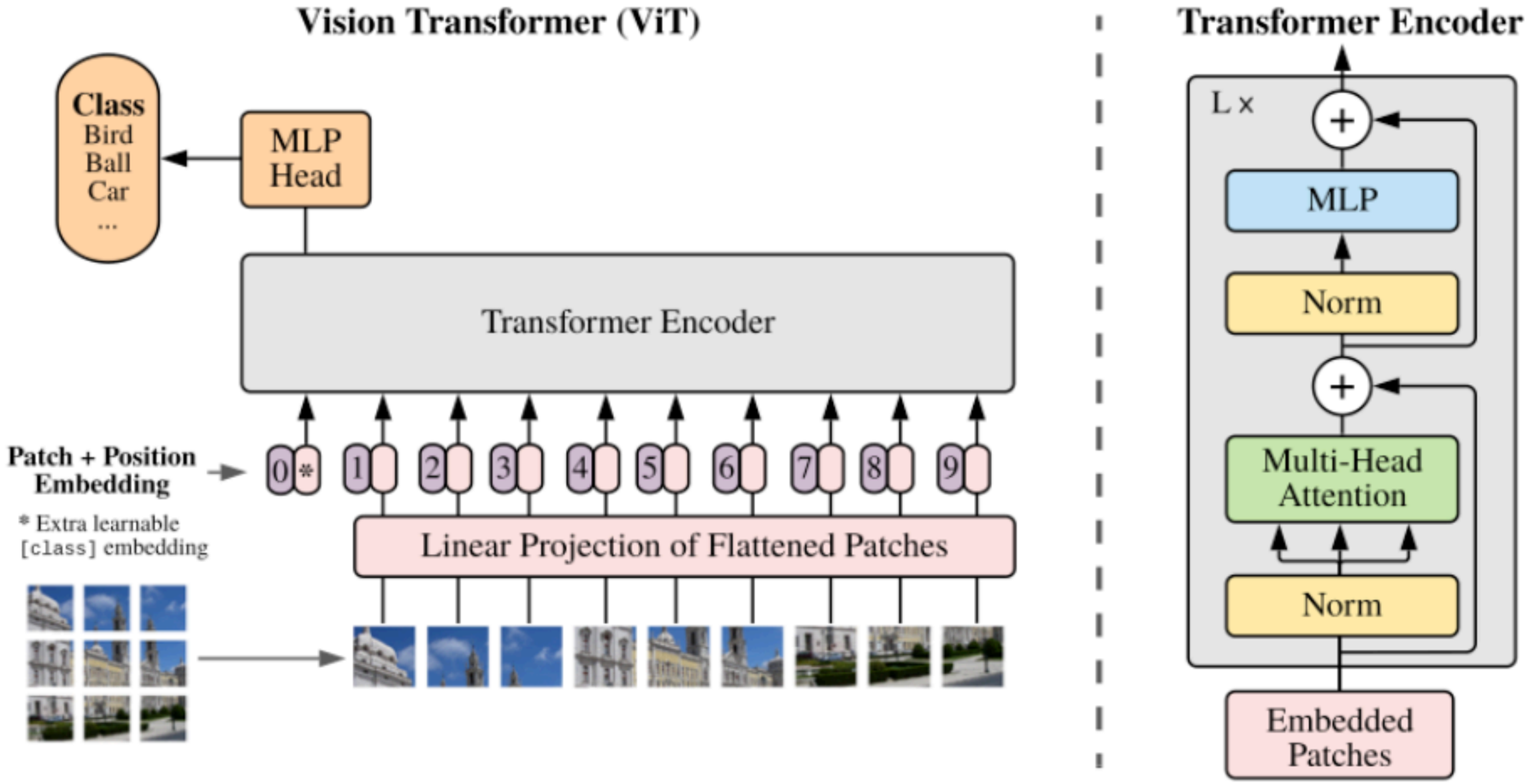
Mix features

1. At a given spatial location
2. Between different spatial locations

Examples:

- **N x N convolutions** (N > 1) , **pooling** implements (2)
- **1 x 1** perform (1)
- **Large kernels** perform both (1) and (2)
- **Self-attention** allows (1) and (2)
- **MLP** does (1)

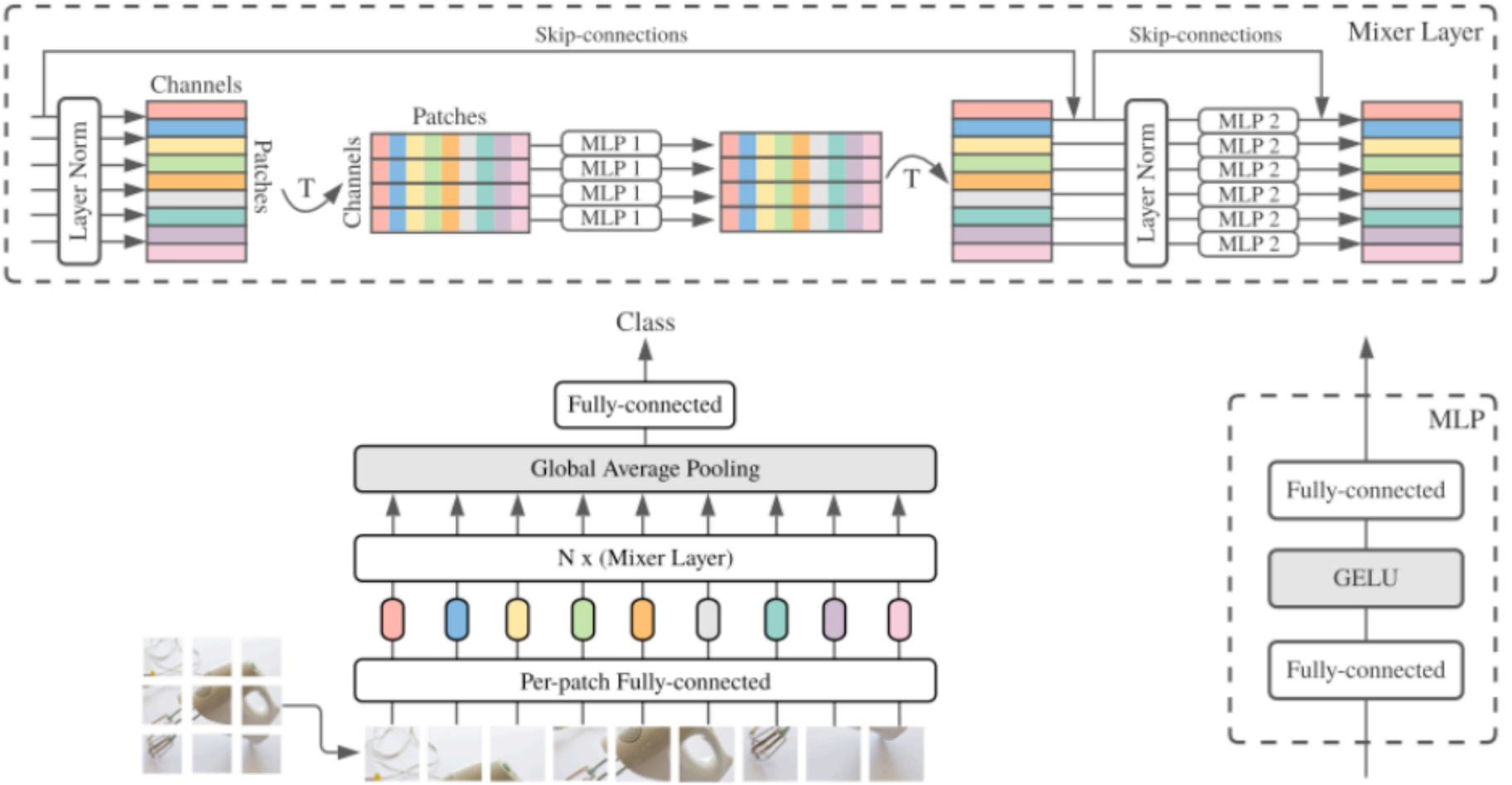
Vision Transformer (ViT)



https://arxiv.org/abs/2010.11929

MLP-Mixer

$$\mathbf{U}_{*,i} = \mathbf{X}_{*,i} + \mathbf{W}_2\,\sigma\big(\mathbf{W}_1\,\text{LayerNorm}(\mathbf{X})_{*,i}\big), \quad \text{for } i = 1 \dots C,$$
$$\mathbf{Y}_{j,*} = \mathbf{U}_{j,*} + \mathbf{W}_4\,\sigma\big(\mathbf{W}_3\,\text{LayerNorm}(\mathbf{U})_{j,*}\big), \quad \text{for } j = 1 \dots S.$$



ResMLP

https://arxiv.org/abs/2105.03404

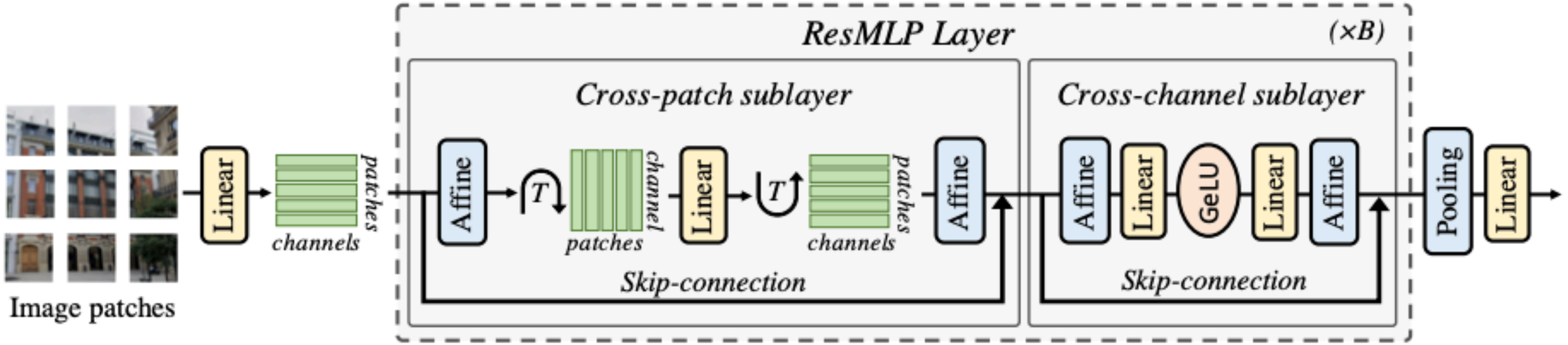
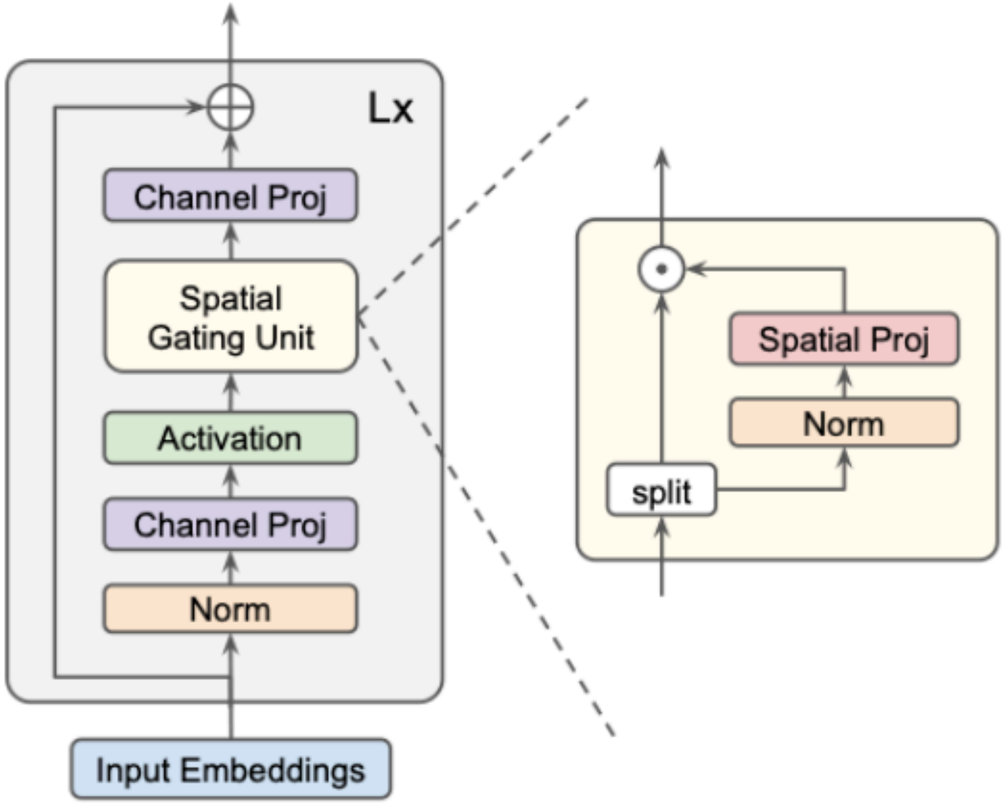


Figure 1: **The ResMLP architecture.** After linearly projecting the image patches into high dimensional embeddings, ResMLP sequentially processes them with (1) a cross-patch linear sublayer; (2) a cross-channel two-layer MLP. The MLP is the same as the FCN sublayer of a Transformer. Each sublayer has a residual connection and two Affine element-wise transformations.

gMLP



Pseudo-code for the gMLP block

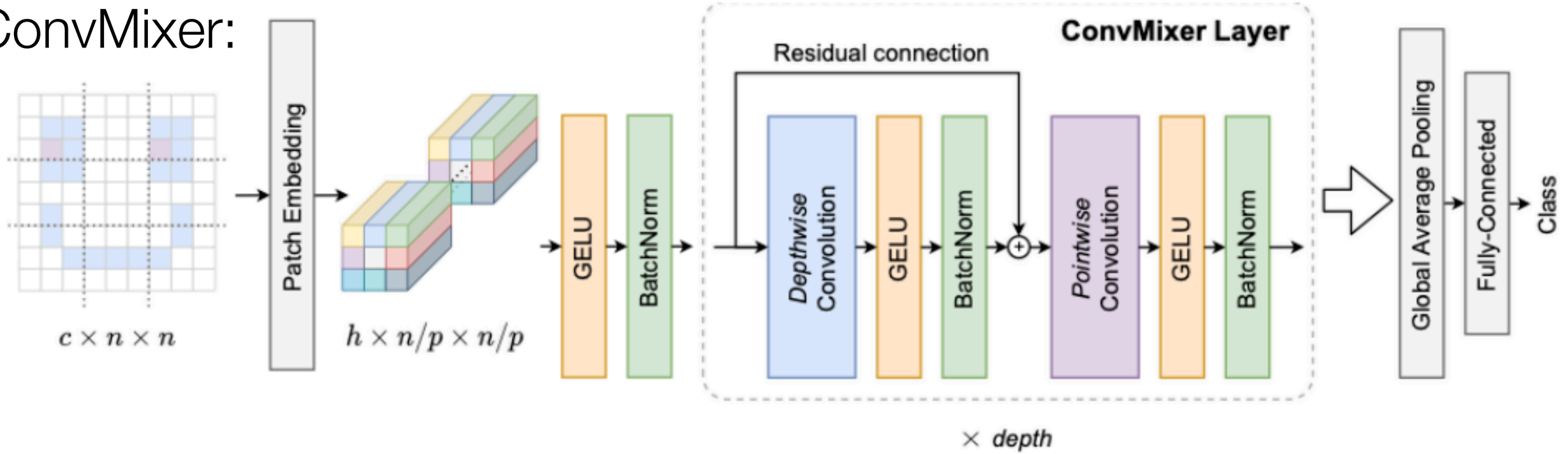
```
def gmlp_block(x, d_model, d_ffn):
  shortcut = x
  x = norm(x, axis="channel")
  x = proj(x, d_ffn, axis="channel")
  x = gelu(x)
  x = spatial_gating_unit(x)
  x = proj(x, d_model, axis="channel")
  return x + shortcut

 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
```

https://arxiv.org/abs/2105.08050v2

~~Convolutions~~ ~~Attention~~ ~~MLPs~~ Patches Are All You Need?

ConvMixer:

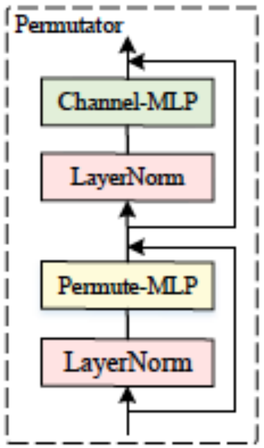


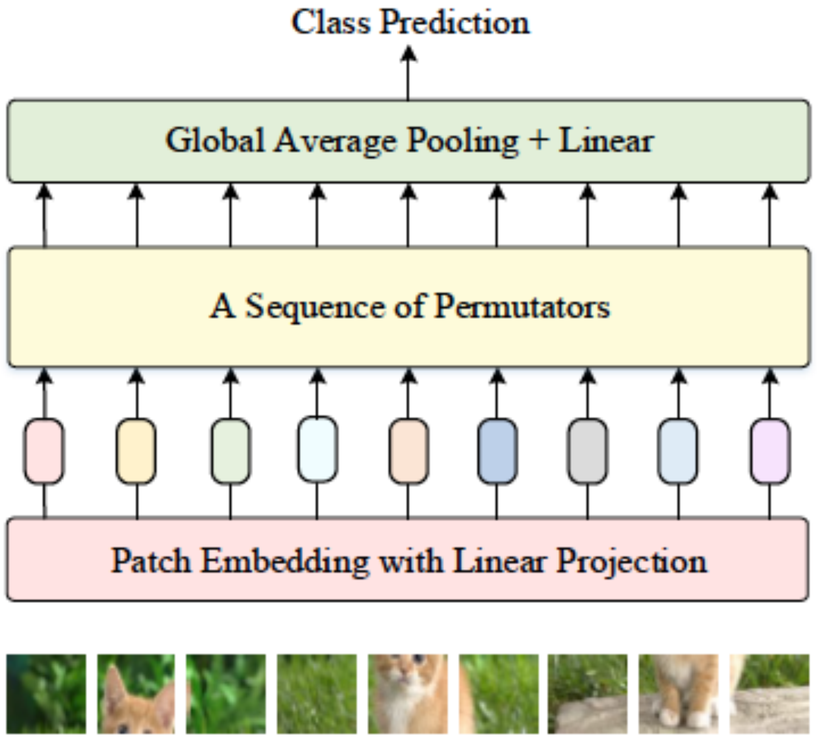

```
def ConvMixer(h, depth, kernel_size=9, patch_size=7, n_classes=1000):
    Seq, ActBn = nn.Sequential, lambda x: Seq(x, nn.GELU(), nn.BatchNorm2d(h))
    Residual = type('Residual', (Seq,), {'forward': lambda self, x: self[0](x) + x})
    return Seq(ActBn(nn.Conv2d(3, h, patch_size, stride=patch_size)),
        *[Seq(Residual(ActBn(nn.Conv2d(h, h, kernel_size, groups=h, padding="same"))),
          ActBn(nn.Conv2d(h, h, 1))) for i in range(depth)],
         nn.AdaptiveAvgPool2d((1,1)), nn.Flatten(), nn.Linear(h, n_classes))
```

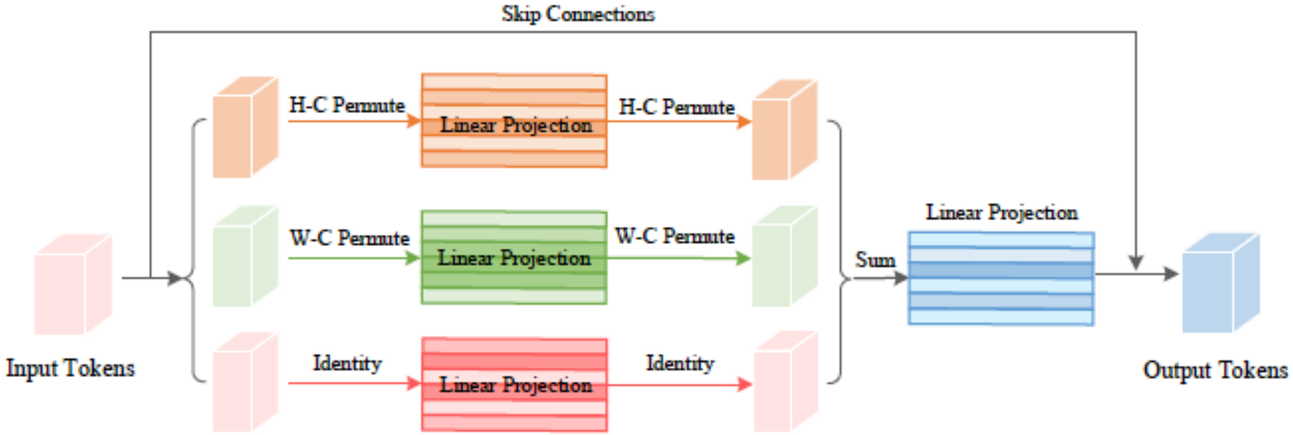
Figure 3: Implementation of ConvMixer in PyTorch; see Appendix D for more implementations.

Under review to ICLR 2022 https://openreview.net/pdf?id=TVHS5Y4dNvM

Vision Permutator

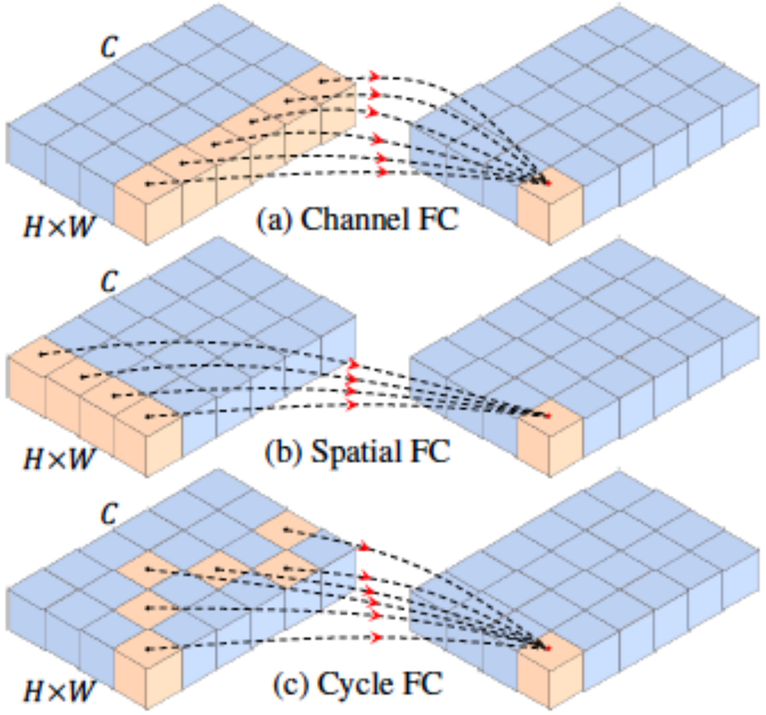




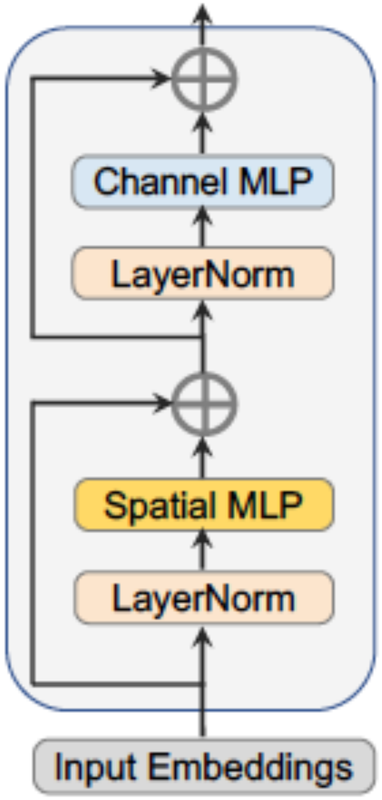


https://arxiv.org/abs/2106.12368

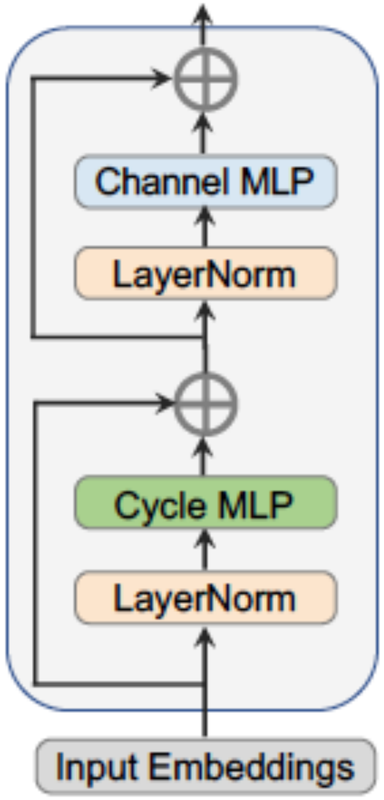
CycleMLP: To cope with variable input image scales



FC	$\mathcal{O}(HW)$	Scale Variable	ImgNet Top-1	COCO AP	ADE20K mIoU
Channel	HW	✓	79.4	35.0	36.3
Spatial	H^2W^2	✗	80.9	✗	✗
Cycle	HW	✓	81.6	41.7	42.4



(a) **MLP-Mixer** [49]



(b) **CycleMLP**

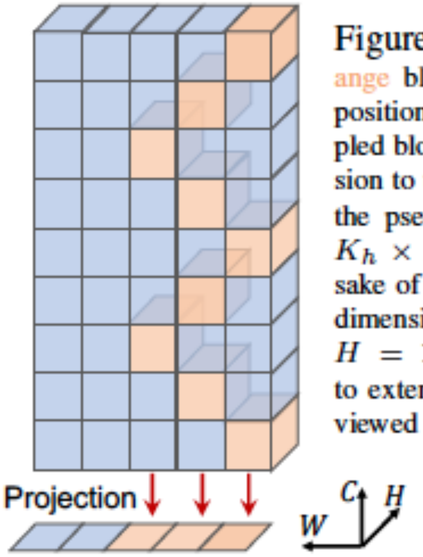


Figure 4: **Pseudo-Kernel:** Orange blocks denote the sampled positions. We project these sampled blocks along channel dimension to the spatial surface and get the pseudo kernel with the size $K_h \times K_w = 1 \times 3$. For the sake of simplicity, we omit batch dimension and take a feature with $H = 1$ for example. It is easy to extend to normal cases. Best viewed in color.

https://arxiv.org/abs/2107.10224

How to train your MLP-Mixer?

Training MLP-Mixer

- Datasets: ILSVRC2021 ImageNet, ImageNet-21k, JFT-300M: de-duplicated and resolution 224x224
- Adam $\beta_1 = 0.9, \beta_2 = 0.999$
- Linear learning rate warmup of 10k steps and linear decay
- Batch size 4096
- Weights decay and gradient clipping at global norm 1
- Special cropping for JFT-300M with random horizontal flipping
- RandAugment, mixup, dropout, stochastic depth
- You can check **timm** library for more details and code: https://fastai.github.io/timmdocs/

Fine-tuning

- SGD
- Batch size 412
- Gradient Clipping at global norm 1
- Cosine learning rate schedule with a linear warmup
- Fine-tune at higher resolutions
- Adjust the first layer with respect to the length of a sequence of tokens
  - By initializing weights by copies of pre-trained ones over diagonal

Experiments & Discussion

Main Results

	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

gMLP on ImageNet

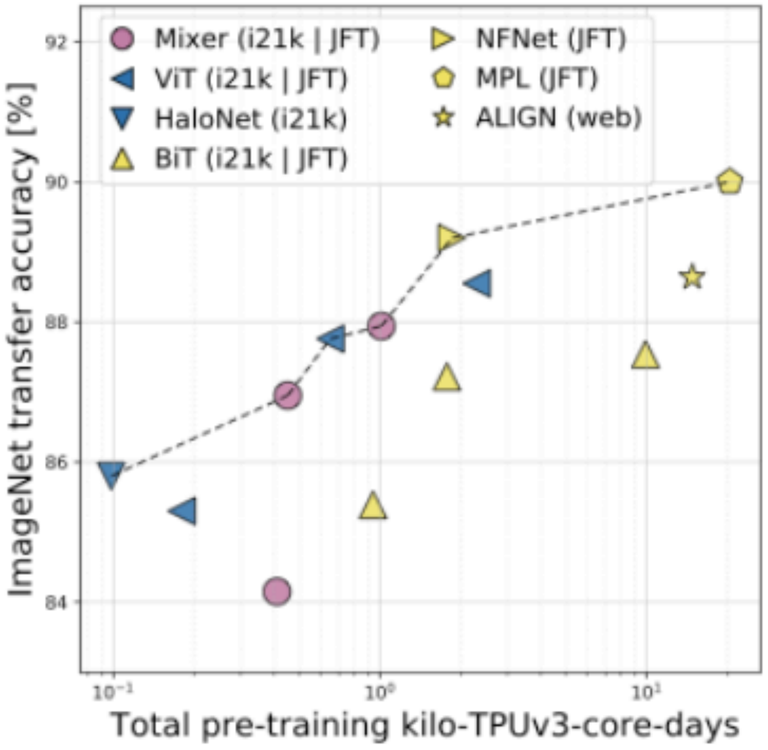
Model	ImageNet Top-1 (%)*	Input Resolution	Params (M)	MAdds (B)
	ConvNets			
ResNet-152 [16]	78.3	224	60	11.3
RegNetY-8GF [39]	81.7	224	39	8.0
EfficientNet-B0 [17]	77.1	224	5	0.39
EfficientNet-B3 [17]	81.6	300	12	1.8
EfficientNet-B7 [17]	84.3	600	66	37.0
NFNet-F0 [33]	83.6	192	72	12.4
	Transformers			
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

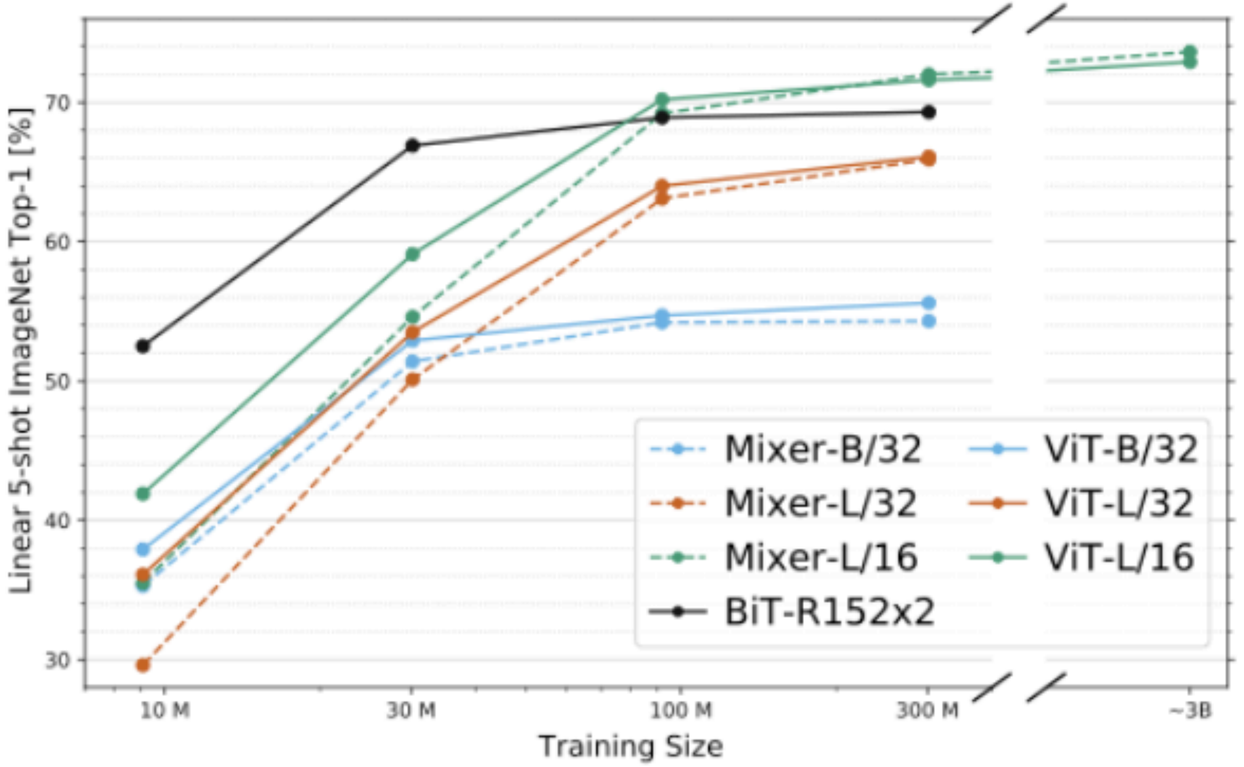
* Standard deviation across multiple independent runs is around 0.1.
† Tokenization & embedding process at the stem can be viewed as a convolution.

ConvMixer on ImageNet

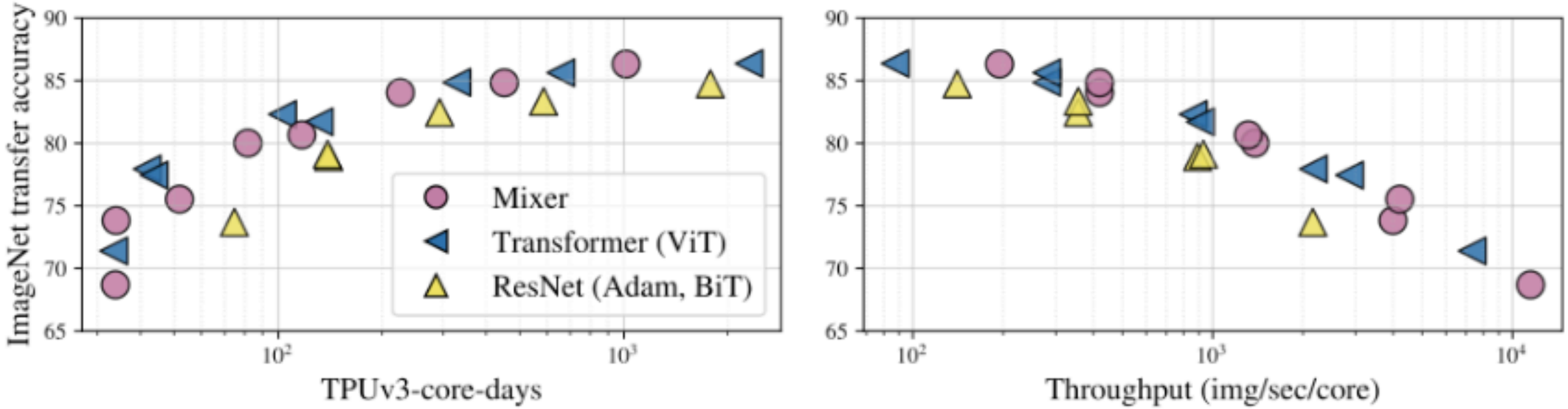
Current "Most Interesting" **ConvMixer** Configurations *vs.* Other Simple Models							
Network	Patch Size	Kernel Size	# Params ($\times 10^6$)	Throughput (img/sec)	Act. Fn.	# Epochs	ImNet top-1 (%)
ConvMixer-1536/20	7	9	51.6	89	G	150	81.37
ConvMixer-768/32	7	7	21.1	203	R	300	80.16
ResNet-152	–	3	60.2	872	R	150	79.64
DeiT-B	16	–	86	703	G	300	81.8
ResMLP-B24/8	8	–	129	140	G	400	81.0

Main Results



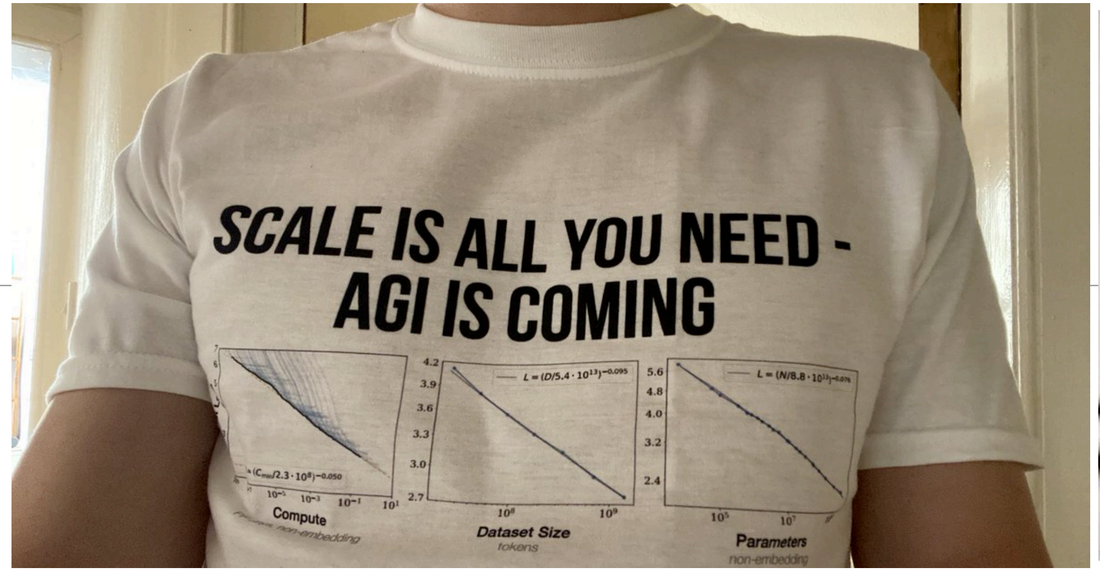


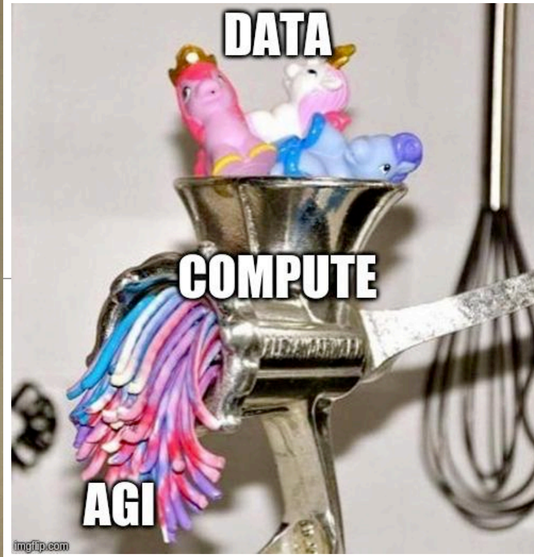
The role of model scale

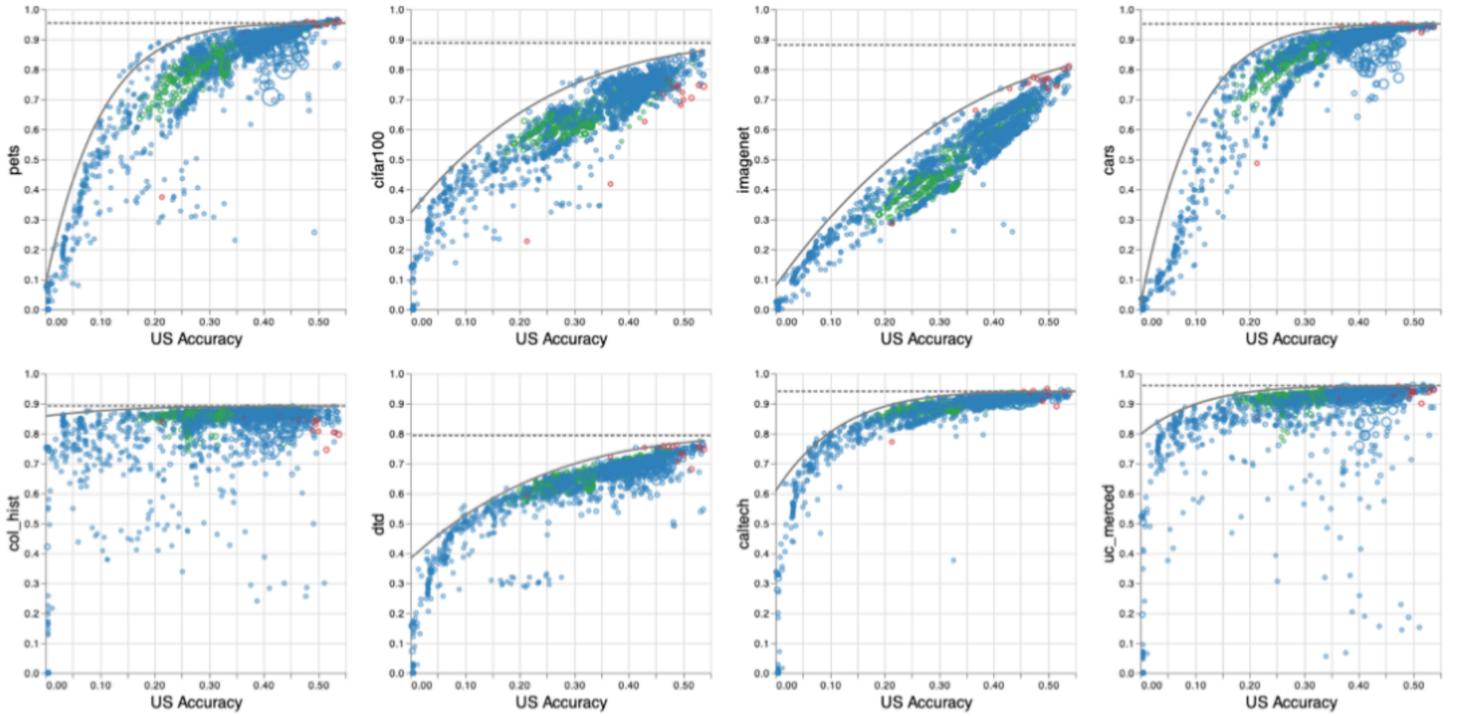


The role of model scale

- Even we improve performance on an upstream task by scaling datasets and models, there is a **saturating behavior** to downstream task
- Scaling does not lead to a one-model-fits-all solution.
- Push information with weight decay and learning rate
  - Lower Layers -> Better Downstream Performance
  - Higher Layers -> Better Upstream Performance.
- Also read **Underspecification** https://arxiv.org/abs/2011.03395v2





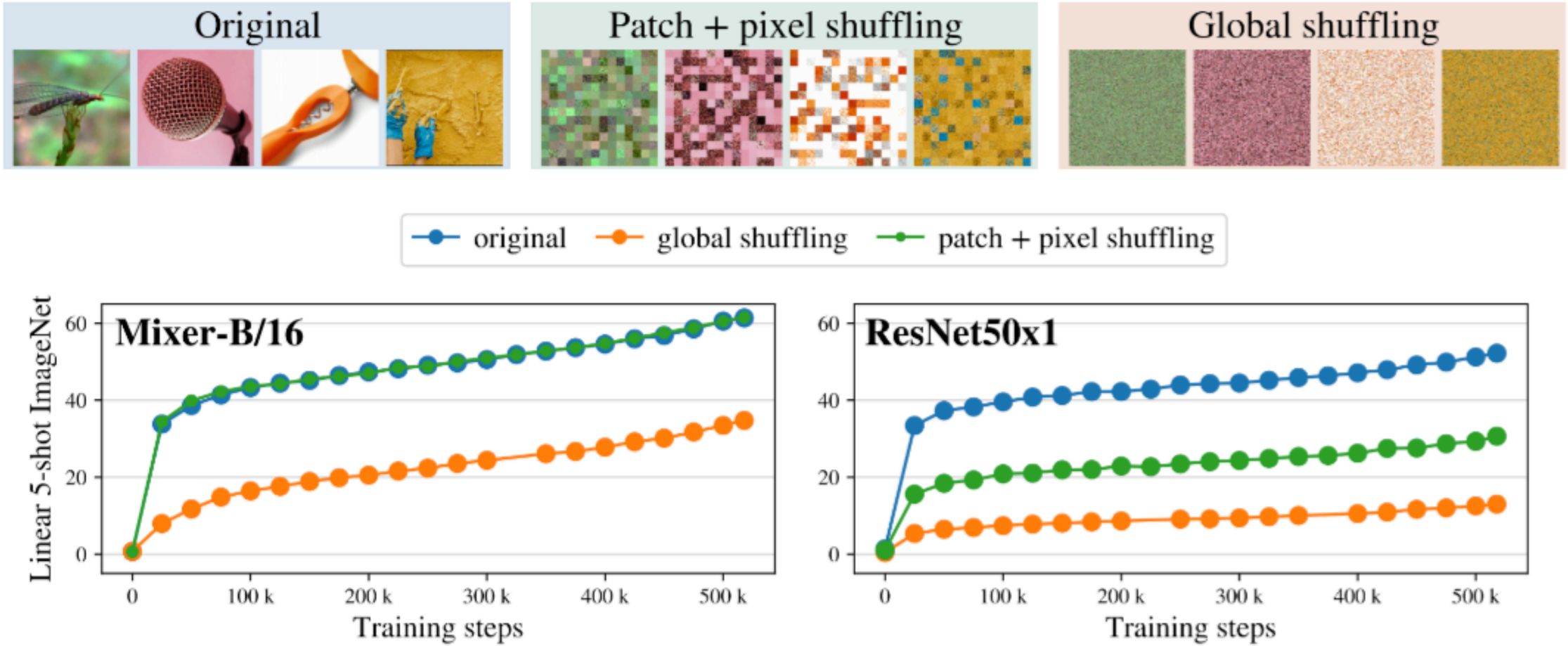


https://arxiv.org/abs/2110.02095

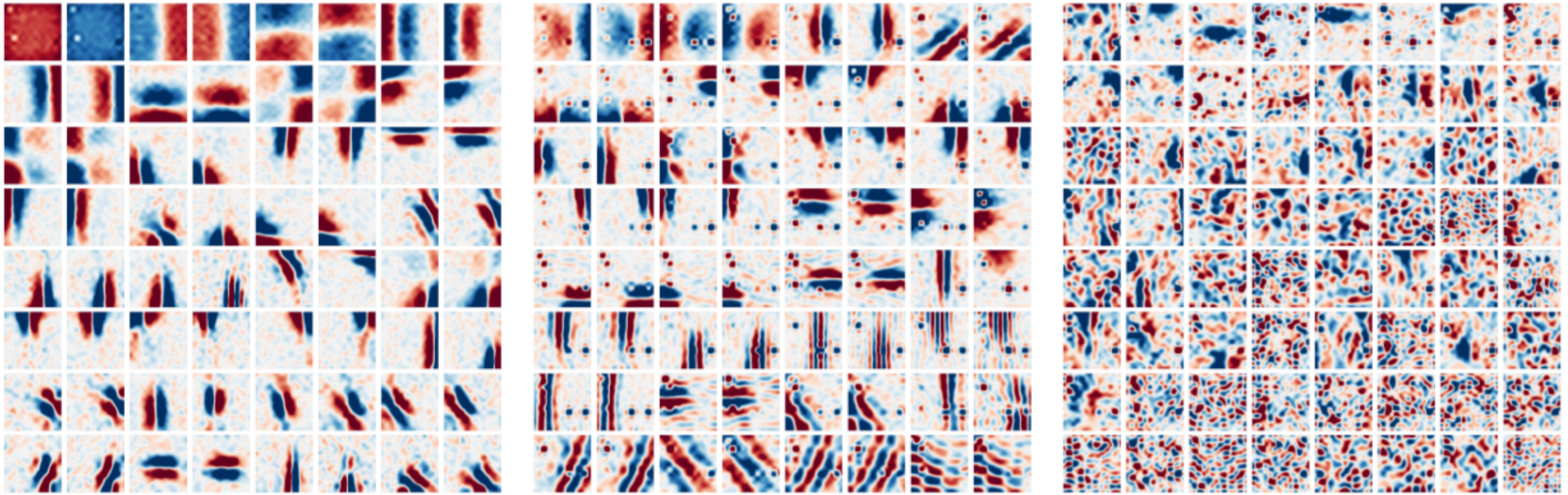
The role of pre-training dataset size

	Image size	Pre-Train Epochs	ImNet top-1	ReaL top-1	Avg. 5 top-1	Throughput (img/sec/core)	TPUv3 core-days
Pre-trained on ImageNet (with extra regularization)							
Mixer-B/16	224	300	76.44	82.36	88.33	1384	0.01k(‡)
ViT-B/16 (☎)	224	300	79.67	84.97	90.79	861	0.02k(‡)
Mixer-L/16	224	300	71.76	77.08	87.25	419	0.04k(‡)
ViT-L/16 (☎)	224	300	76.11	80.93	89.66	280	0.05k(‡)
Pre-trained on ImageNet-21k (with extra regularization)							
Mixer-B/16	224	300	80.64	85.80	92.50	1384	0.15k(‡)
ViT-B/16 (☎)	224	300	84.59	88.93	94.16	861	0.18k(‡)
Mixer-L/16	224	300	82.89	87.54	93.63	419	0.41k(‡)
ViT-L/16 (☎)	224	300	84.46	88.35	94.49	280	0.55k(‡)
Mixer-L/16	448	300	83.91	87.75	93.86	105	0.41k(‡)
Pre-trained on JFT-300M							
Mixer-S/32	224	5	68.70	75.83	87.13	11489	0.01k
Mixer-B/32	224	7	75.53	81.94	90.99	4208	0.05k
Mixer-S/16	224	5	73.83	80.60	89.50	3994	0.03k
BiT-R50x1	224	7	73.69	81.92	—	2159	0.08k
Mixer-B/16	224	7	80.00	85.56	92.60	1384	0.08k
Mixer-L/32	224	7	80.67	85.62	93.24	1314	0.12k
BiT-R152x1	224	7	79.12	86.12	—	932	0.14k
BiT-R50x2	224	7	78.92	86.06	—	890	0.14k
BiT-R152x2	224	14	83.34	88.90	—	356	0.58k
Mixer-L/16	224	7	84.05	88.14	94.51	419	0.23k
Mixer-L/16	224	14	84.82	88.48	94.77	419	0.45k
ViT-L/16	224	14	85.63	89.16	95.21	280	0.65k
Mixer-H/14	224	14	86.32	89.14	95.49	194	1.01k
BiT-R200x3	224	14	84.73	89.58	—	141	1.78k
Mixer-L/16	448	14	86.78	89.72	95.13	105	0.45k
ViT-H/14	224	14	86.65	89.56	95.57	87	2.30k
ViT-L/16 [14]	512	14	87.76	90.54	95.63	32	0.65k

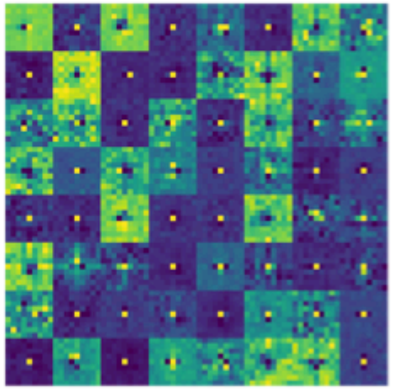
Invaruance to input permutations



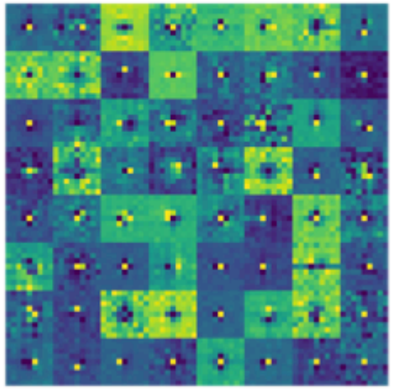
Visualization



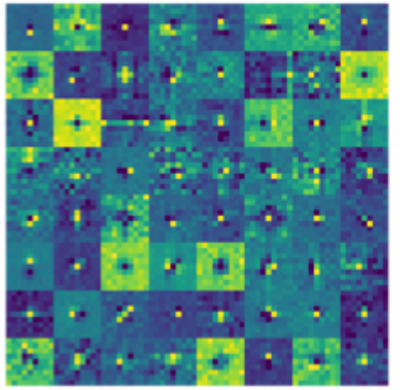
ConvMixer



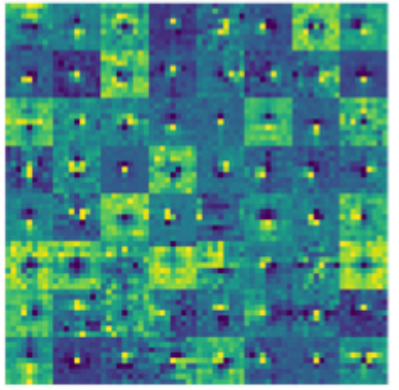
(a) Layer 1



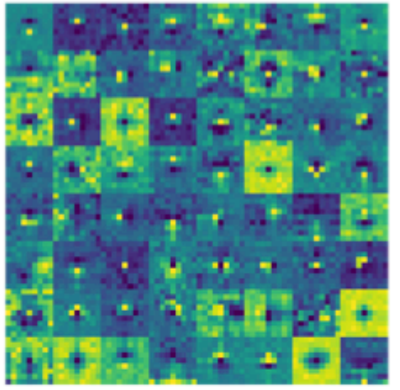
(b) Layer 4



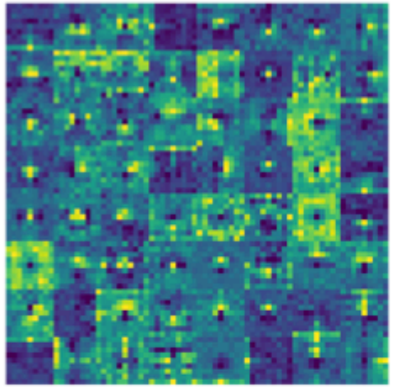
(c) Layer 6



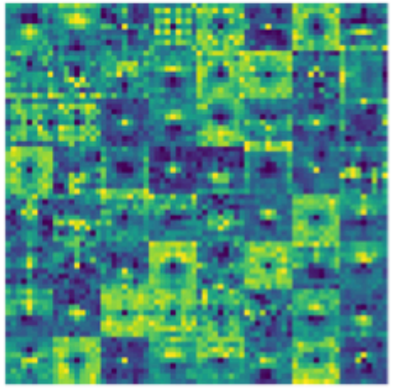
(d) Layer 9



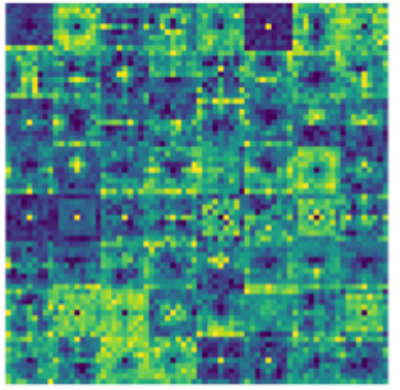
(e) Layer 11



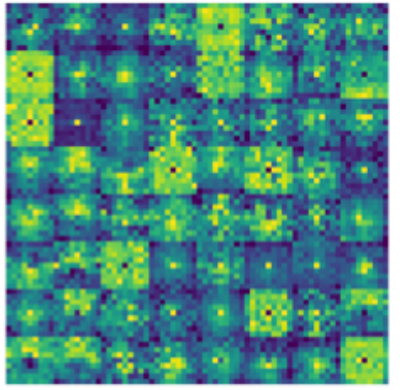
(f) Layer 13



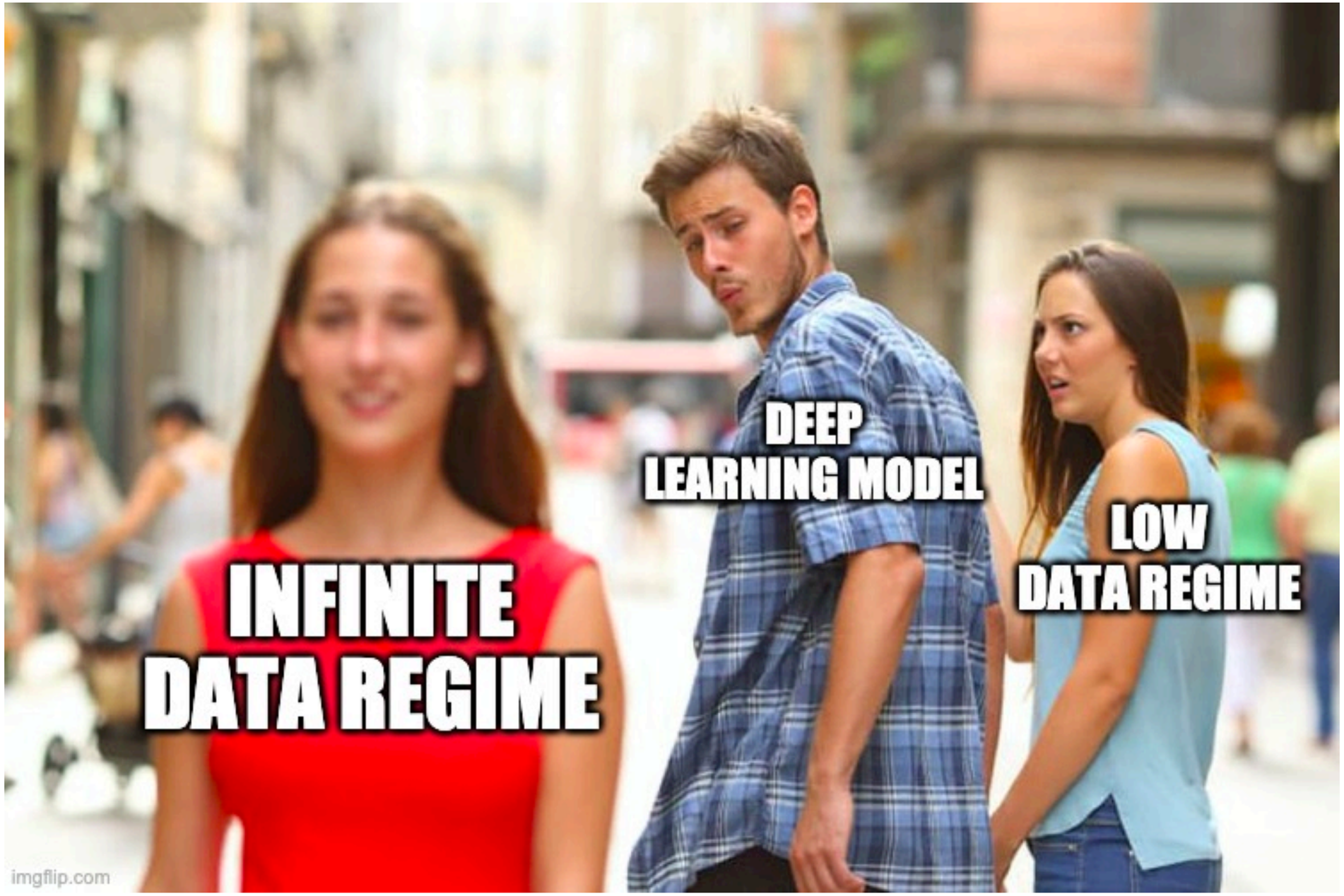
(g) Layer 15



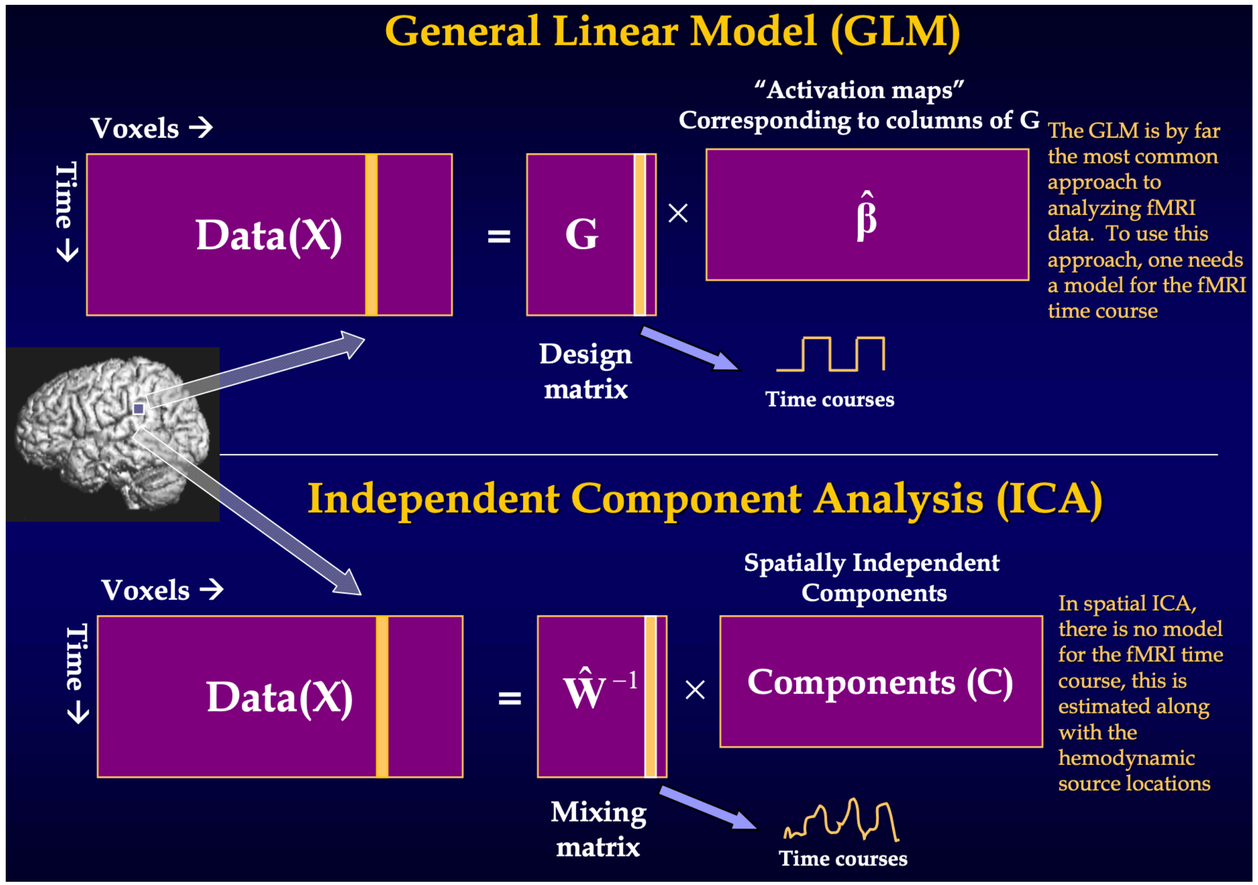
(h) Layer 17

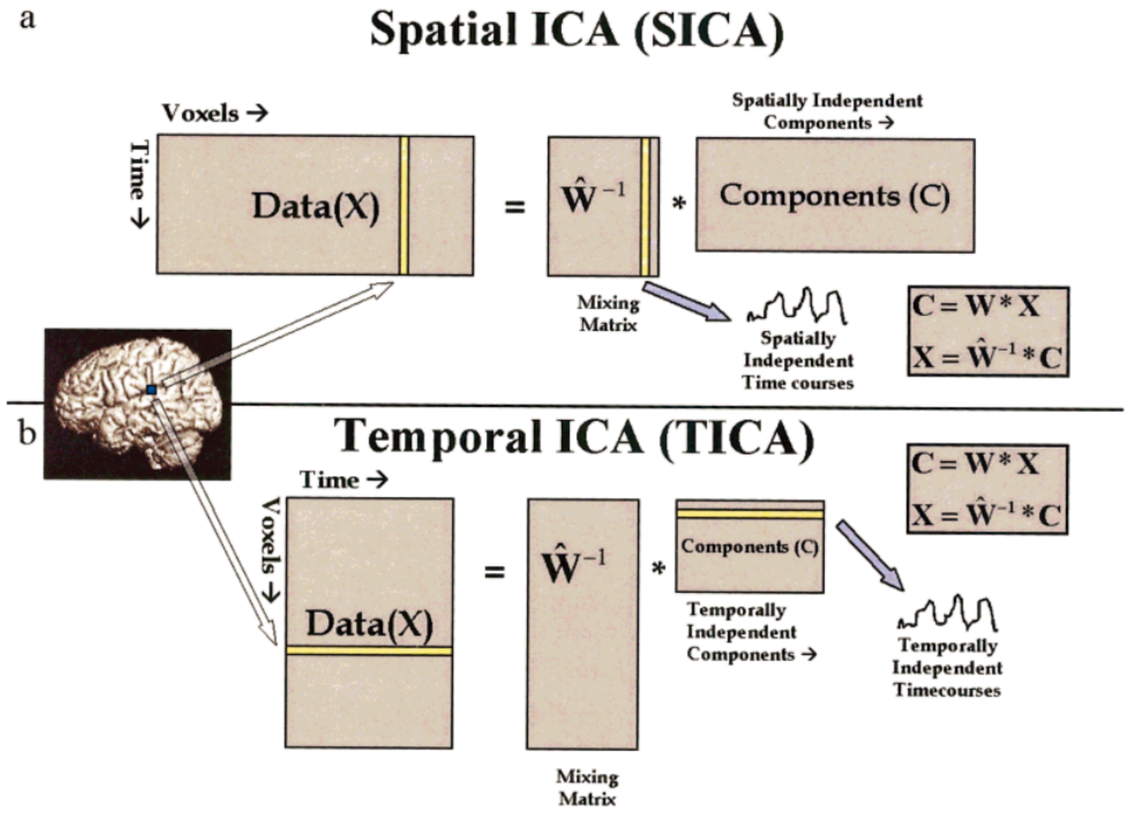


(i) Layer 19


INFINITE
DATA REGIME
DEEP
LEARNING MODEL
LOW
DATA REGIME
imgflip.com

http://helper.ipam.ucla.edu/publications/mbi2008/mbi2008_7652.pdf





https://onlinelibrary.wiley.com/doi/epdf/10.1002/hbm.1024