Re-Analysis of the Traditional Architecture

Neural ODE, ResNet Strikes Back, Patches Are All You Need

2021. 11. 4

Dajin Han

Index

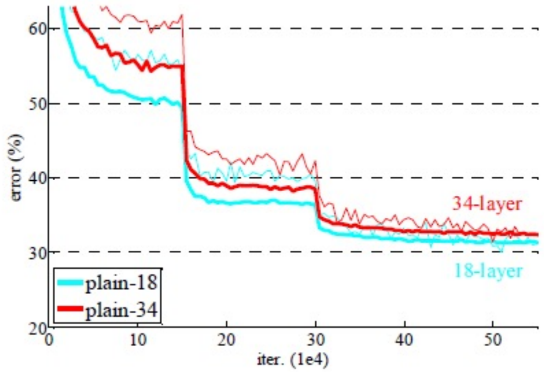
- ResNet
  - Analysis on Neural ODE
  - ResNet Strikes Back
- Vision Transformer
  - Patches Are All You Need

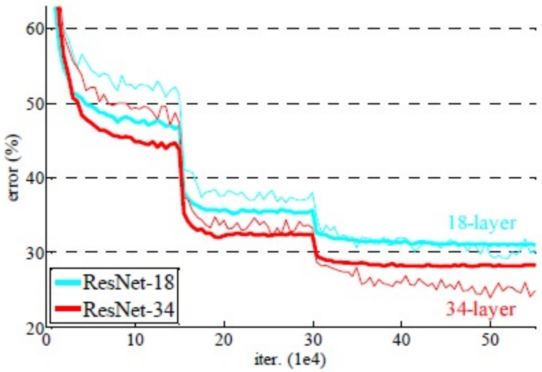
Neural ODE

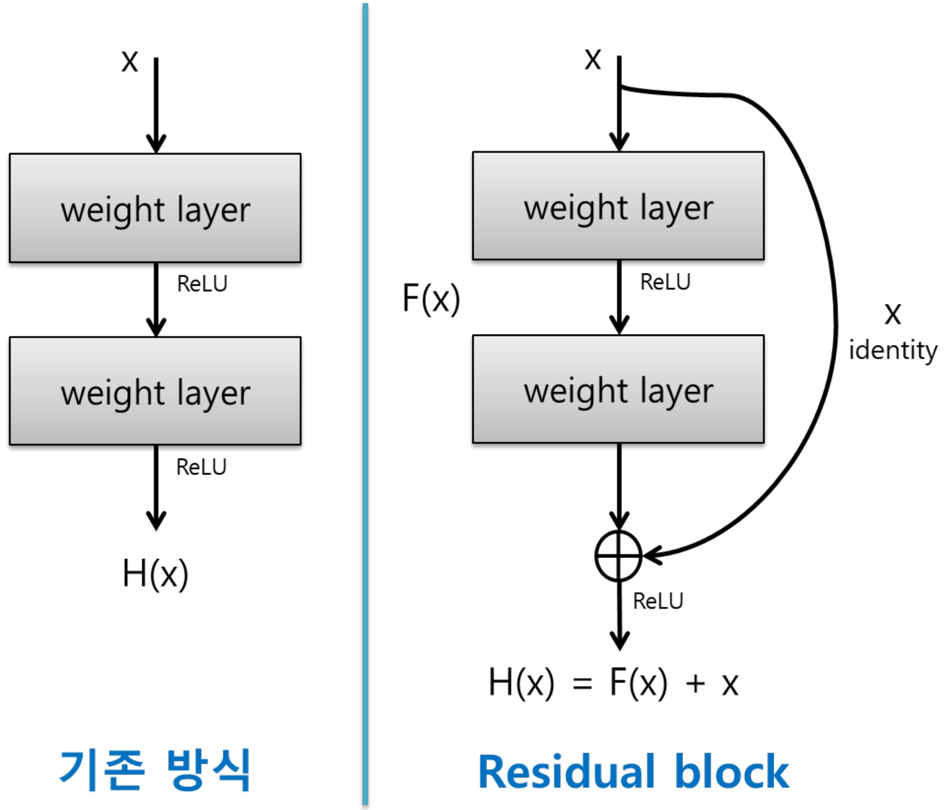
Similarity of Residual Block and Euler's Method

ResNet

- Residual block
- Shorcut / Skip connection
- Enable deeper network
  - Plane layer : reproduced info
  - Residual block : additional info, if necessary







Euler's Method

- Taylor series $$T_f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \frac{1}{6}f'''(a)(x-a)^3 + \cdots$$

- Euler's Method $$y(t_{i+1}) = y(t_i) + (t_{i+1} - t_i)y'(t_i) + \frac{(t_{i+1} - t_i)^2}{2}y''(\xi_i)$$
  - Solve Differential Equation

Similarity

ResNet: $$x_{k+1} = x_k + F(x_k)$$

Euler's Method: $$x_{k+1} = x_k + hF(x_k),$$

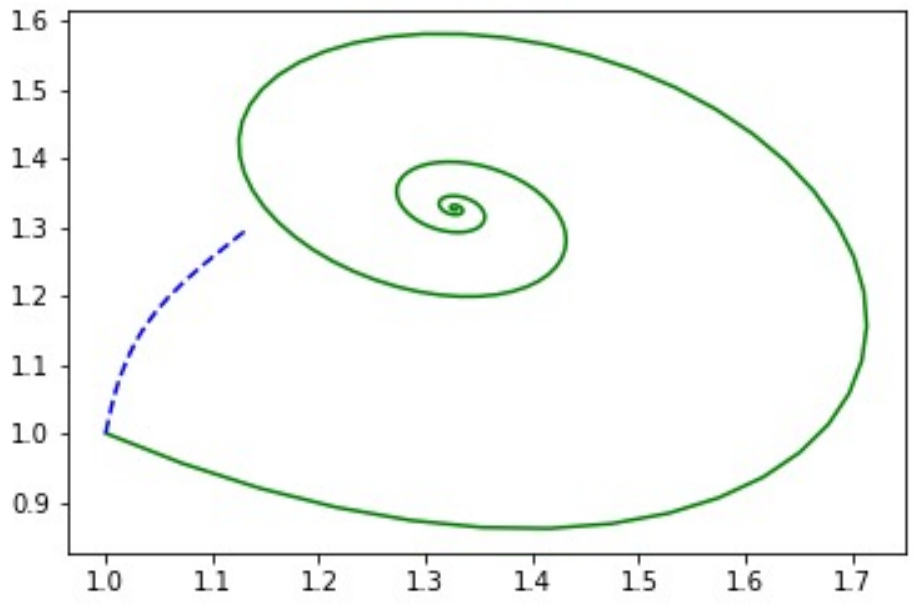
- Solve Differential Equation
  - From initial point to the target point
  - Initial point == Input Image
  - Target point == prediction

Neural ODE

- Continuous Model
  - No Fixed(Discrete) Layer

- Ordinary Differential Equation
  - For single function, single variable

$$\frac{dy}{dt} = f\left(t, y\right) \qquad y\left(t_0\right) = y_0$$

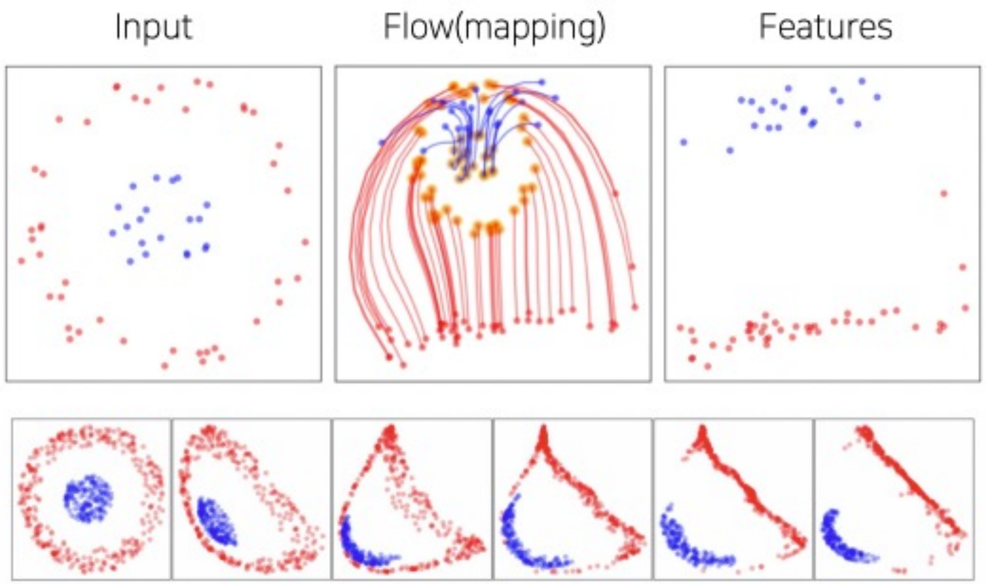


Neural ODE

- Continuous Model
  - No Fixed(Discrete) Layer

- Ordinary Differential Equation
  - For single function, single variable

$$\frac{dy}{dt} = f(t, y) \qquad y(t_0) = y_0$$

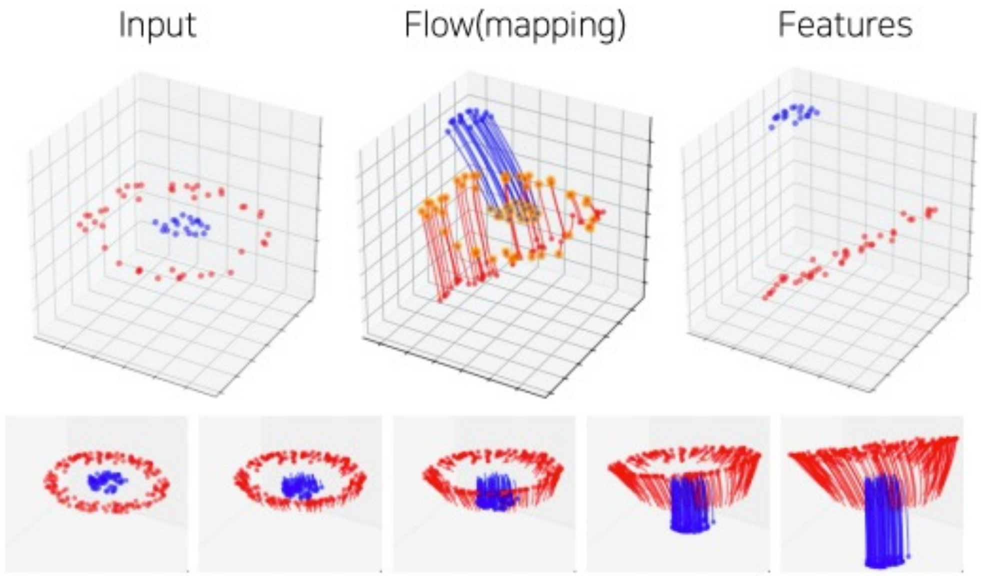


Neural ODE

- Continuous Model
  - No Fixed(Discrete) Layer

- Ordinary Differential Equation
  - For single function, single variable

$$\frac{dy}{dt} = f(t, y) \qquad y(t_0) = y_0$$

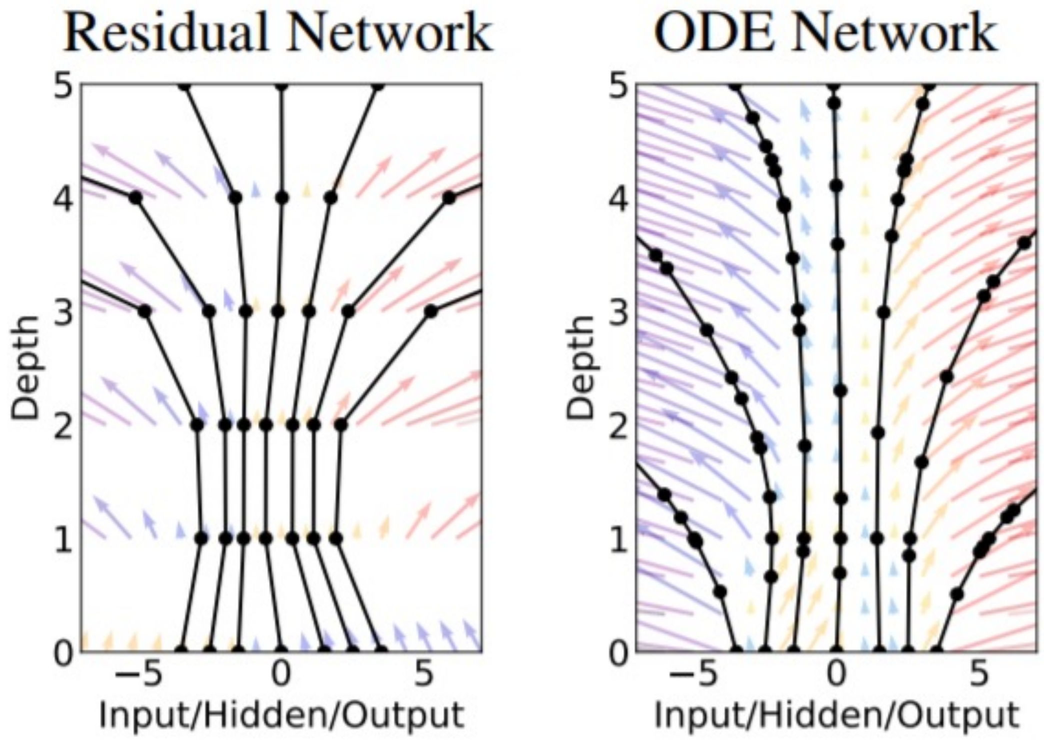


Similarity

- Solve Differential Equation
  - From initial point to the target point
  - Initial point == Input Image
  - Target point == prediction

ResNet: $x_{k+1} = x_k + F(x_k)$

Euler's Method: $x_{k+1} = x_k + hF(x_k),$



ResNet Strikes Back

Timm, Advanced Pretrained ResNet, Fair Comparison

Model Accuracy

$$\text{accuracy (model)} = f(\mathcal{A}, \mathcal{T}, \mathcal{N}),$$

- Architecture design
- Training setting
- Measurement noise

Training Procedures

Procedure A1 aims at providing the best performance for ResNet-50. It is therefore the longest in terms of epochs (600) and training time (4.6 days on one node with 4 V100 32GB GPUs).

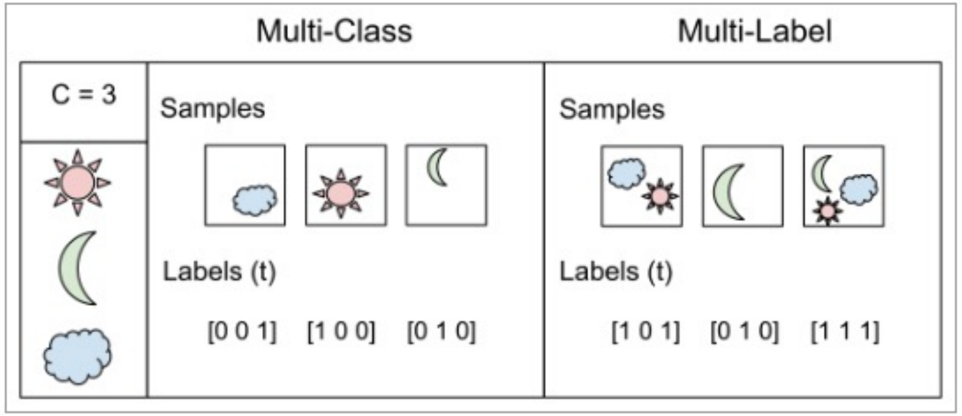
Procedure A2 is a 300 epochs schedule that is comparable to several modern procedures like DeiT, except with a larger batch size of 2048 and other choices introduced for all our recipes.

Procedure A3 aims at outperforming the original ResNet-50 procedure with a short schedule of 100 epochs and a batch size 2048. It can be trained in 15h on 4 V100 16GB GPUs and could be a good setting for exploratory research or studies.

Training Procedure	Number of epochs	Training resolution	Training time	Peak memory by GPU (MB)	Numbers of GPU	Top-1 accuracy val	real	v2
A1	600	224×224	110h	22,095	4	80.4	85.7	68.7
A2	300	224×224	55h	22,095	4	79.8	85.4	67.9
A3	100	160×160	15h	11,390	4	78.1	84.5	66.1

Training Procedures

- Loss
  - Multi-Label Classification
  - BCE
- Data Augmentation
  - Timm
  - Mixup, Cutmix
- Regularization
  - Weight Decay
  - Label smoothing
  - Repeated Augmentation
  - Stochastic-Depth
- Optimization
  - LAMB
    - Normalize gradient and scale it by layer

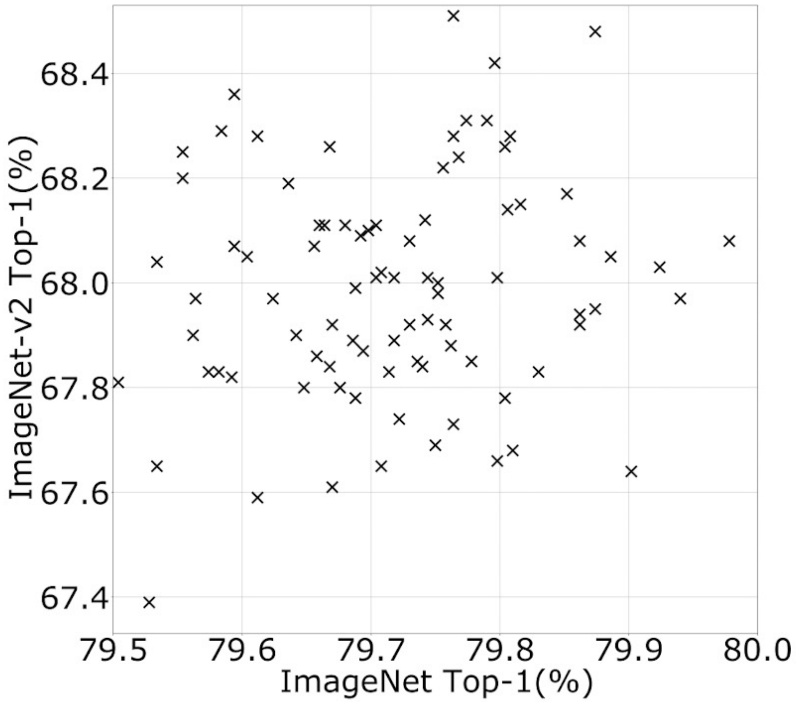


Experiments

	A1-A2-org.		A3		Cost							ImageNet-1k-val			
	train	test	train	test	A1	A2	A1-A2		A3			A1	A2	A3	org.
↓ Architecture	res.	res.	res.	res.	time (hour)		# GPU	Pmem	time	# GPU	Pmem	Accuracy(%)			
ResNet-18 [13]†	224	224	160	224	186	93	2	12.5	28	2	6.5	71.5	70.6	68.2	69.8
ResNet-34 [13]†	224	224	160	224	186	93	2	17.5	27	2	9.0	76.4	75.5	73.0	73.3
ResNet-50 [13]†	224	224	160	224	110	55	4	22.0	15	4	11.4	80.4	79.8	78.1	76.1
ResNet-101 [13]†	224	224	160	224	74	37	8	16.3	8	8	8.5	81.5	81.3	79.8	77.4
ResNet-152 [13]†	224	224	160	224	92	46	8	22.5	9	8	11.8	82.0	81.8	80.6	78.3
RegNetY-4GF [32]	224	224	160	224	130	65	4	27.1	15	4	13.9	81.5	81.3	79.0	79.4
RegNetY-8GF [32]	224	224	160	224	106	53	8	19.8	10	8	10.3	82.2	82.1	81.1	79.9
RegNetY-16GF [32]	224	224	160	224	150	75	8	25.6	13	8	13.4	82.0	82.2	81.7	80.4
RegNetY-32GF [32]	224	224	160	224	120	60	16	17.6	12	16	9.4	82.5	82.4	82.6	81.0
SE-ResNet-50 [20]	224	224	160	224	102	51	4	27.6	16	4	14.2	80.0	80.1	77.0	76.7
SENet-154 [20]	224	224	160	224	110	55	16	23.3	12	16	12.2	81.7	81.8	81.9	81.3
ResNet-50-D [14]	224	224	160	224	100	50	4	23.9	14	4	12.3	80.7	80.2	78.7	79.3
ResNeXt-50-32x4d [51]†	224	224	160	224	80	40	8	14.3	15	4	14.6	80.5	80.4	79.2	77.6
EfficientNet-B0 [41]	224	224	160	224	110	55	4	22.1	15	4	11.4	77.0	76.8	73.0	77.1
EfficientNet-B1 [41]	240	240	160	224	62	31	8	17.9	8	8	7.9	79.2	79.4	74.9	79.1
EfficientNet-B2 [41]	260	260	192	256	76	38	8	22.8	9	8	11.9	80.4	80.1	77.5	80.1
EfficientNet-B3 [41]	300	300	224	288	62	31	16	19.5	6	16	10.1	81.4	81.4	79.2	81.6
EfficientNet-B4 [41]	380	380	320	380	64	32	32	20.4	8	32	14.3	81.6	82.4	81.2	82.9
ViT-Ti [45]*	224	224	160	224	98	49	4	16.3	14	4	7.0	74.7	74.1	66.7	72.2
ViT-S [45]*	224	224	160	224	68	34	8	16.1	8	8	7.0	80.6	79.6	73.8	79.8
ViT-B [11]*	224	224	160	224	66	33	16	16.4	5	16	7.3	80.4	79.8	76.0	81.8
timm [50] specific architectures															
ECA-ResNet-50-T	224	224	160	224	112	56	4	29.3	15	4	15.0	81.3	80.9	79.6	-
EfficientNetV2-rw-S [42]	288	384	224	288	52	26	16	16.6	7	16	10.1	82.3	82.9	80.9	83.8
EfficientNetV2-rw-M [42]	320	384	256	352	64	32	32	18.5	9	32	12.1	80.6	81.9	82.3	84.8
ECA-ResNet-269-D	320	416	256	320	108	54	32	27.4	11	32	17.8	83.3	83.9	83.3	85.0

Experiments

- Not Fair Comparison
  - For Dataset
  - For Model



	Top-1 accuracy (%)				
dataset ↓	mean	std	max	min	seed 0
ImageNet-val	79.72	0.10	79.98	79.50	79.85
ImageNet-real	85.37	0.08	85.55	85.21	85.45
ImageNet-V2	67.99	0.23	68.69	67.39	67.90

Figure 1: *Top* ↑: Statistics for ResNet-50 trained with A2 and 100 different seeds. The column "seed 0" corresponds to the weights that we take as reference. Its performance is +0.13% above the average top-1 accuracy on Imagenet-val.

← *Left:* Point cloud plotting the ImageNet-val top-1 accuracy vs ImageNet-V2 for all seeds. Note that the outlying seed that achieves 68.5% top-1 accuracy on ImageNet-V2 has an average performance on ImageNet-val.

Experiments

- Not Fair Comparison
  - For Dataset
  - For Model

Dataset	Train size	Test size	#classes	Pytorch [1]	A1	A2	A3
ImageNet-val [36]	1,281,167	50,000	1000	76.1	**80.4**	79.8	78.1
iNaturalist 2019 [18]	265,240	3,003	1,010	73.2	73.9	**75.0**	73.8
Flowers-102 [29]	2,040	6,149	102	**97.9**	**97.9**	**97.9**	97.5
Stanford Cars [24]	8,144	8,041	196	92.5	**92.7**	92.6	92.5
CIFAR-100 [25]	50,000	10,000	100	86.6	**86.9**	86.2	85.3
CIFAR-10 [25]	50,000	10,000	10	98.2	**98.3**	98.0	97.6

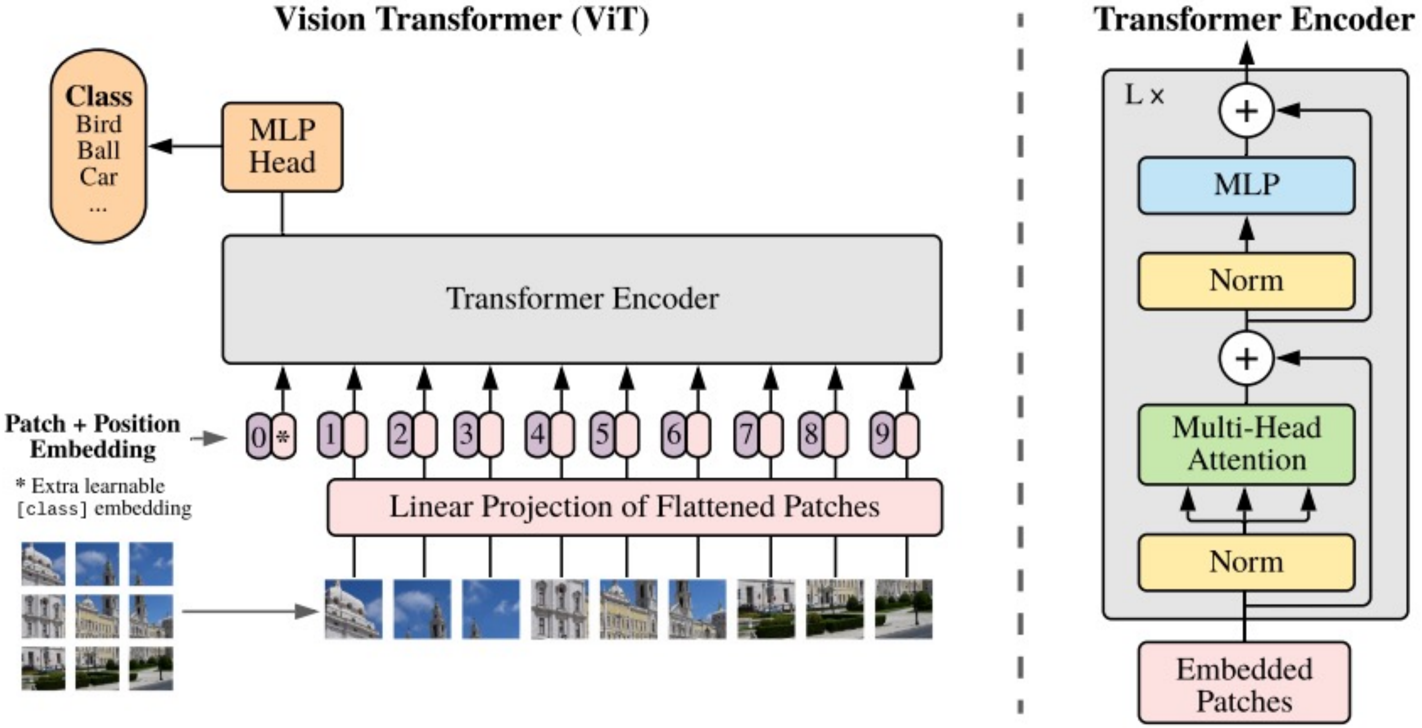
	test set →	ImageNet-val		ImageNet-v2	
↓ architecture	training →	A2	T2	A2	T2
ResNet-50		79.9	79.2	67.9	67.9
DeiT-S		79.6	80.4	68.1	69.2

Patches Are All You Need

Attention, Patch

Vision Transformer

- Patch + Positional Embedding
- **Self-Attention**



Patches Are All You Need

- Traditional CNN
  - Image Input
  - CNN Architecture
- ViT
  - Patch Input
  - Transformer Architecture
- PAAYN
  - Patch Input
  - CNN Architecture

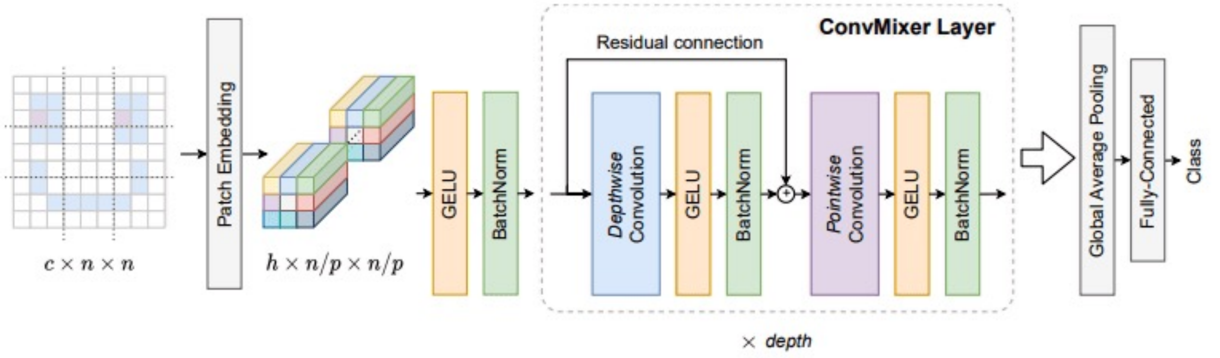
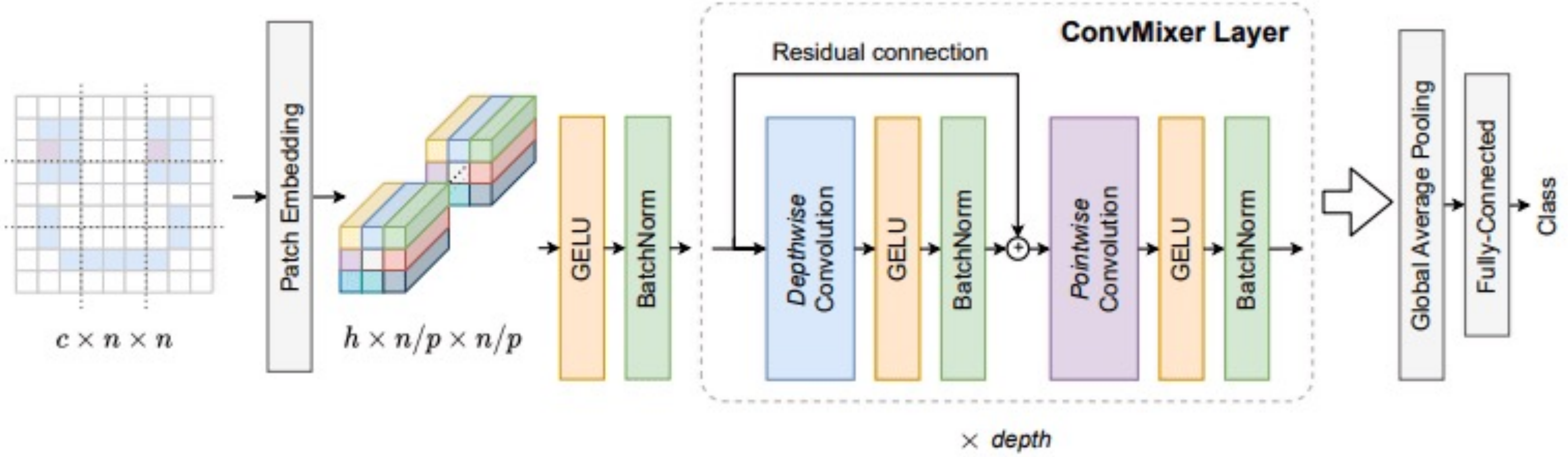


Figure 2: ConvMixer uses "tensor layout" patch embeddings to preserve locality, and then applies d copies of a simple fully-convolutional block consisting of *large-kernel* depthwise convolution followed by pointwise convolution, before finishing with global pooling and a simple linear classifier.

```
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))
```

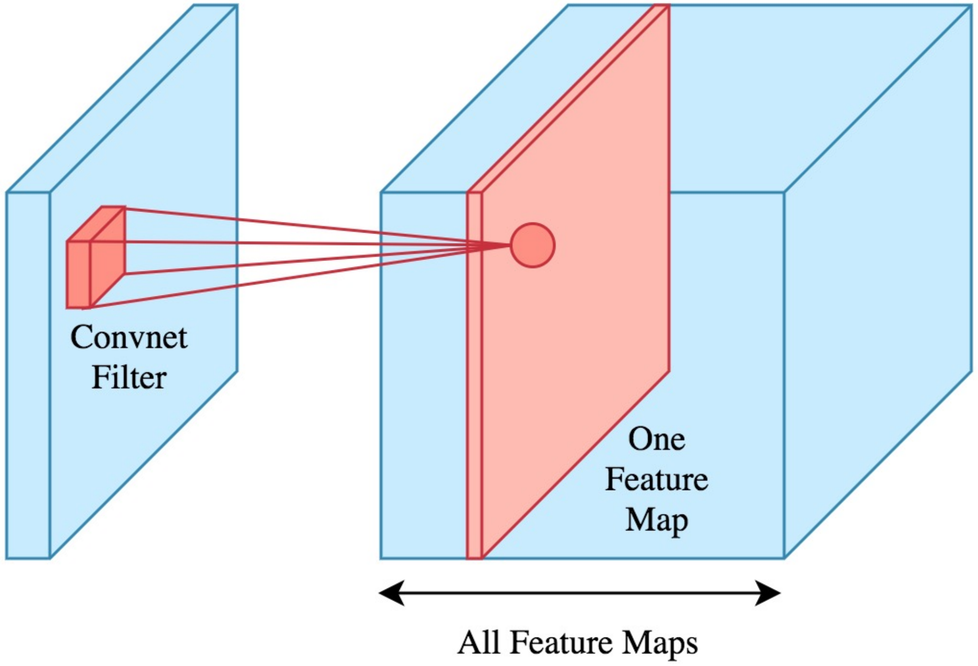
Figure 3: Implementation of ConvMixer in PyTorch; see Appendix D for more implementations.

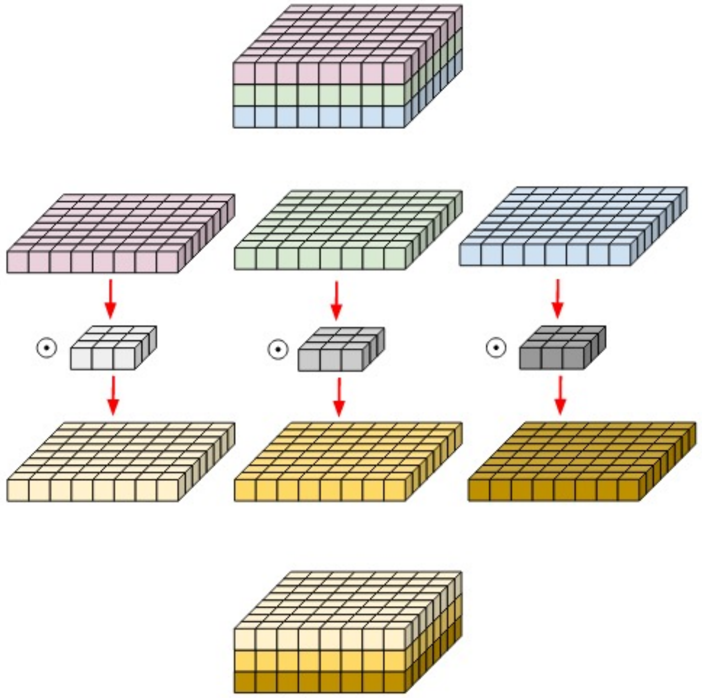
Patches Are All You Need



Convolutions

- Standard Convolution
- Depthwise Convolution





GELU

$$\mathrm{GELU}(x) = xP(X \le x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}(x/\sqrt{2})\right],$$

- Gaussian Error Linear Unit
  - ReLU - Deterministic
  - Dropout – Stochastic
- Good
  - Bounded below
  - Non-monotonic
  - Unbounded above
  - Smooth

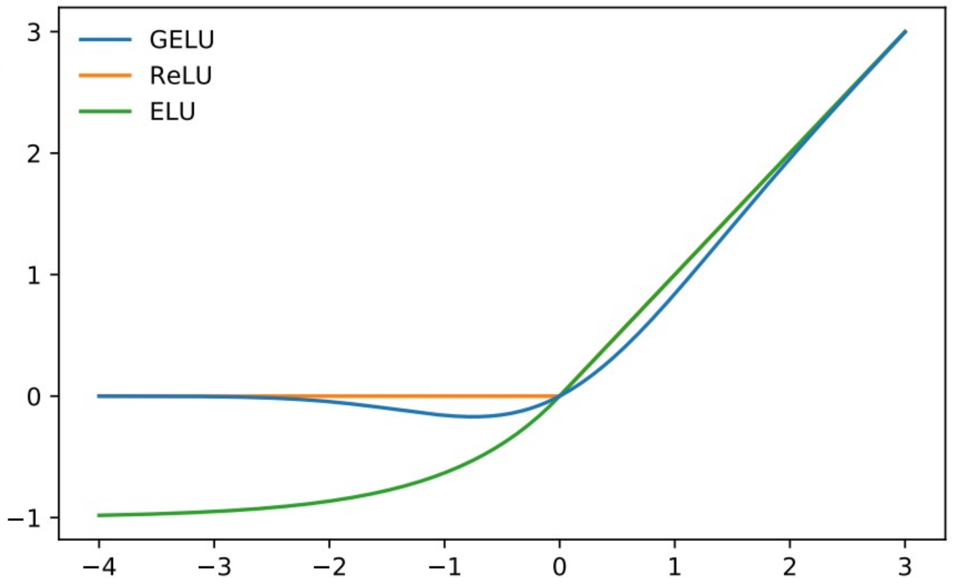


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Experiments

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

Table 1: Models trained and evaluated on 224×224 ImageNet-1k only. See more in Appendix A.

QA