NEW ARCHITECTURES AND TRICKS TO TRAIN THEM

Deep Learning for Computer Vision

Arthur Douillard

https://arthurdouillard.com/deepcourse

Neural Architecture Search

Amoeba

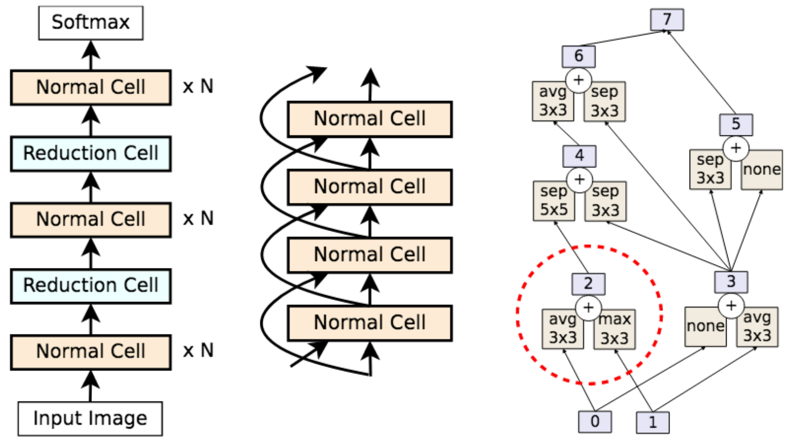


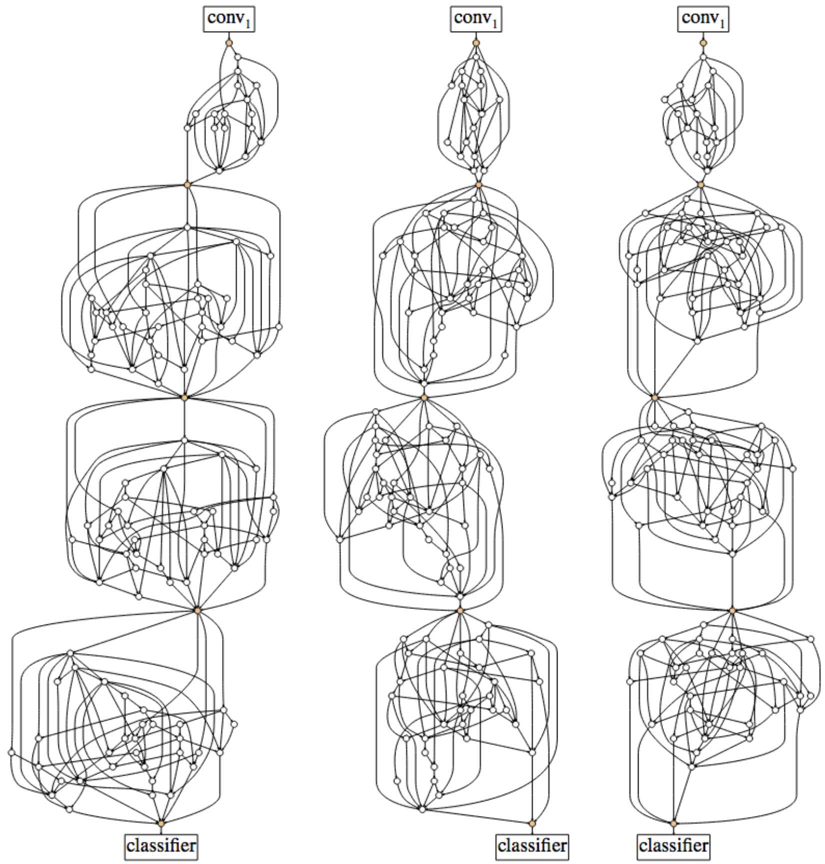
Figure 1: NASNet Search Space [54]. LEFT: the full outer structure (omitting skip inputs for clarity). MIDDLE: detailed view with the skip inputs. RIGHT: cell example. Dotted line demarcates a pairwise combination.

- Given a fixed architecture (left & middle), learn to find the optimal cell (right)
- Learning is done here with an **evolutionary algorithm** that needs to retrain & check model accuracy FOR EACH new mutation!

- NAS are usually very computation intensive, and thus it's mostly big private lab that works on it. With Quoc Le's team at Google Brain the main one.

[Real et al. AAAI 2019]

RandWire



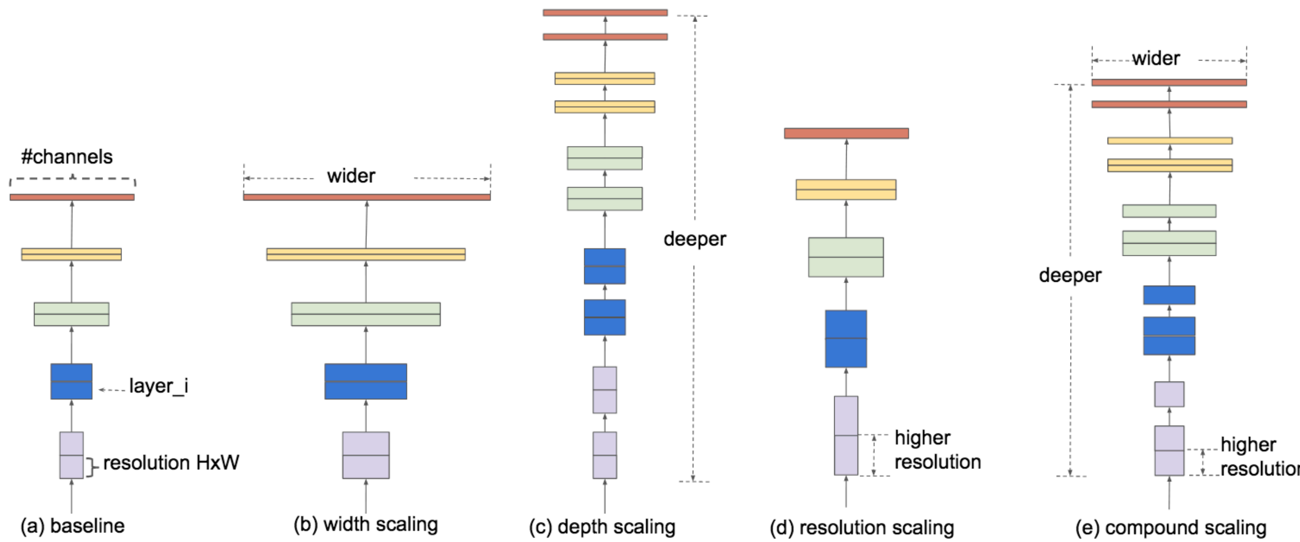
- Or try randomly wired neural network based on minimal set of rules
- But note that some approaches use reinforcement learning

[Xie et al. arXiv 2019]

EfficientNet

$$
\begin{aligned}
\max_{d,w,r} \quad & Accuracy\big(\mathcal{N}(d,w,r)\big) \\
s.t. \quad & \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\big(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\big) \\
& \text{Memory}(\mathcal{N}) \leq \text{target\_memory} \\
& \text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}
\end{aligned}
$$

$$
\begin{aligned}
\text{depth: } & d = \alpha^{\phi} \\
\text{width: } & w = \beta^{\phi} \\
\text{resolution: } & r = \gamma^{\phi} \\
\text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
& \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}
$$



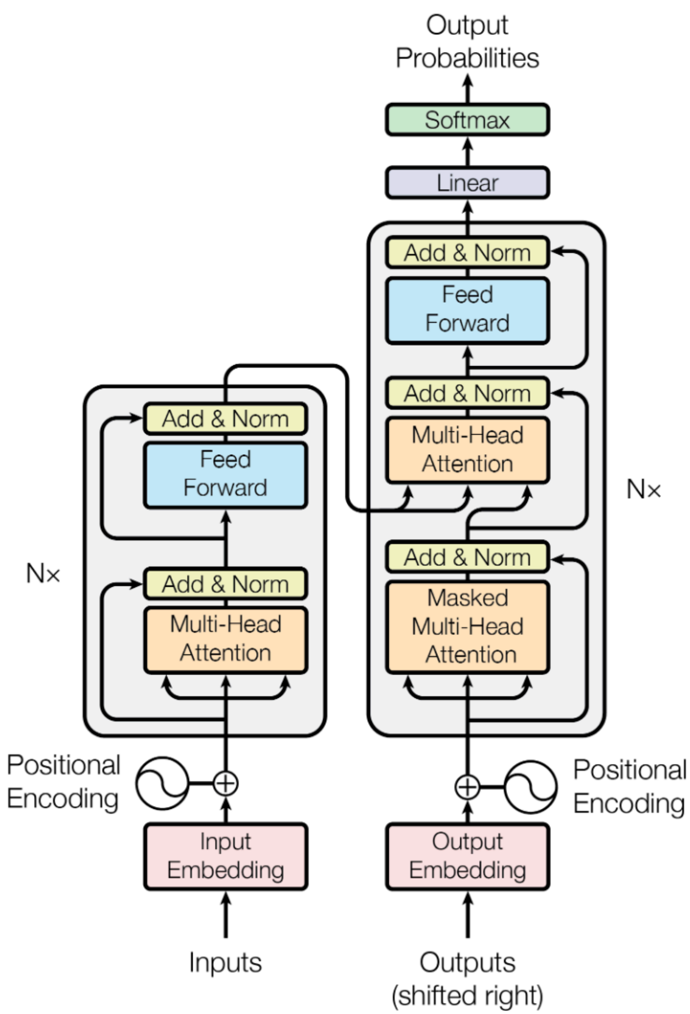
- EfficientNet, one of the best ConvNet as of 2021, was made with NAS
- Based on a **compound scaling** rule they drastically reduce the space to grid search

[Tan and Le, ICML 2019]

Transformers

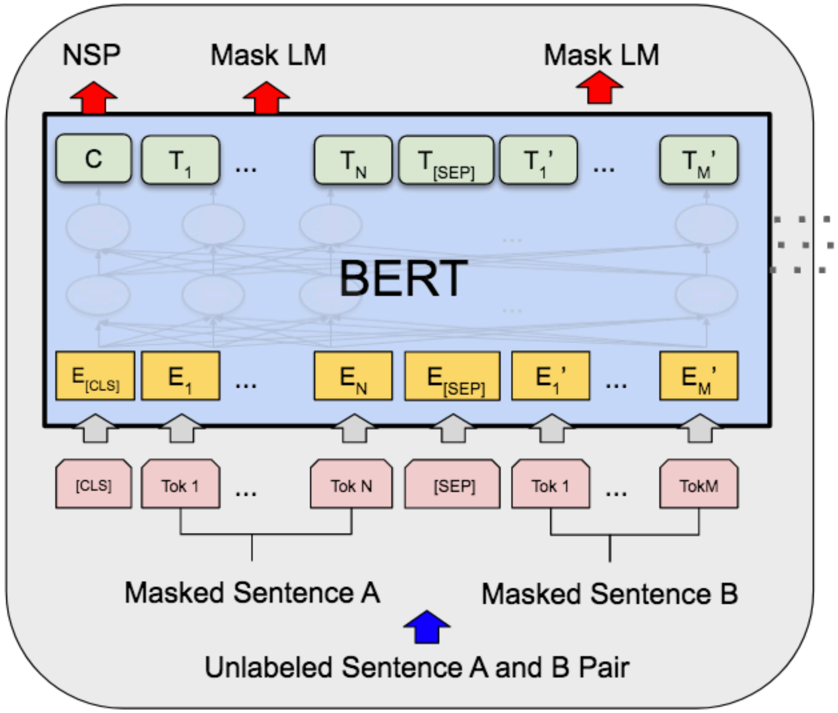
Attention is all you need

- At first designed for machine translation
- Does not rely on Conv1d or RNN
- Main block are FC layers and the famous **Multi-Head Attention**
- Made of a encoder (left) and decoder (right)



[Tan and Le, ICML 2019]

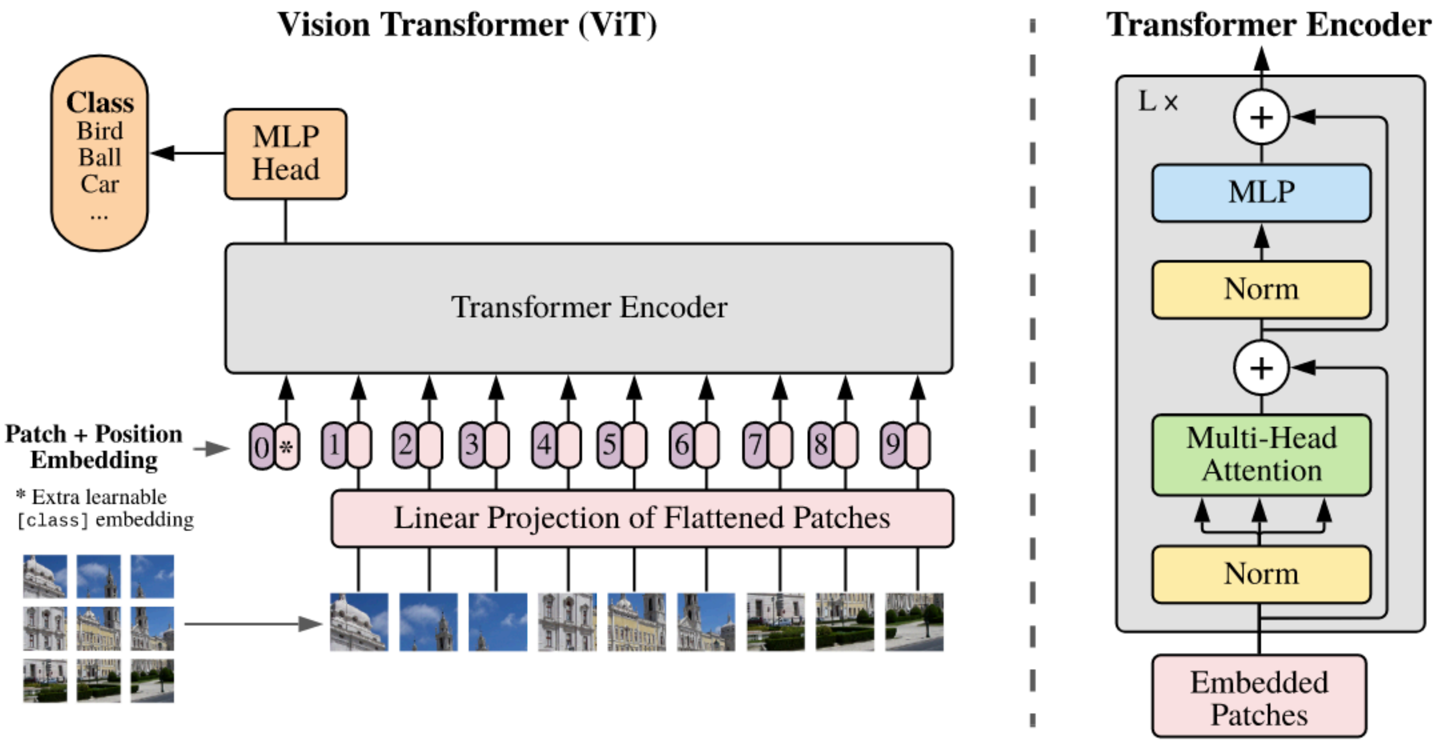
BERT



- Modification of the Transformer
- No more encoder/decoder, only many blocks
- Introduction of a special token [CLS] that is learned
- Was a big revolution in the NLP field

[Devlin et al. NAACL 2019]

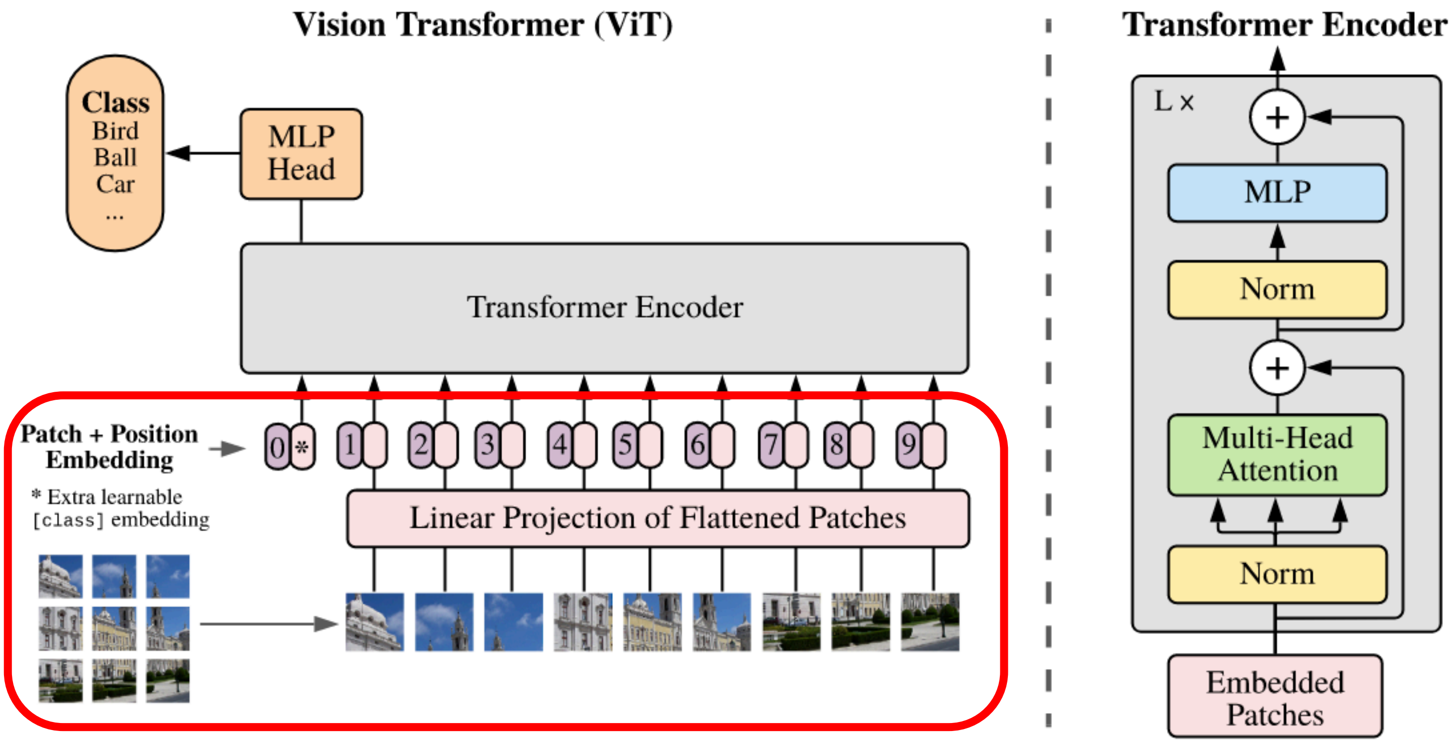
ViT



- First *full* application of transformers with a BERT-like architecture to vision

[Dosovitskiy, Beyer, Kolesnikov et al. ICLR 2021]

Patch, Position, and class token



- Use a convolution with a kernel size equal to the patch size to generate the **tokens**
  - Total size is thus *(batch size, number of tokens, embedding dimension)*
- Add an **extra token [class]** that is a learned vector of size *(embedding dimension)*
- Add to all tokens a learned **positional embeddings**

[Dosovitskiy, Beyer, Kolesnikov et al. ICLR 2021]

Self-Attention

1. Apply three different linear transformations, to create the **Q**uery, the **K**ey, and the **V**alue

$$Q = XW_q$$

$$K = XW_k$$

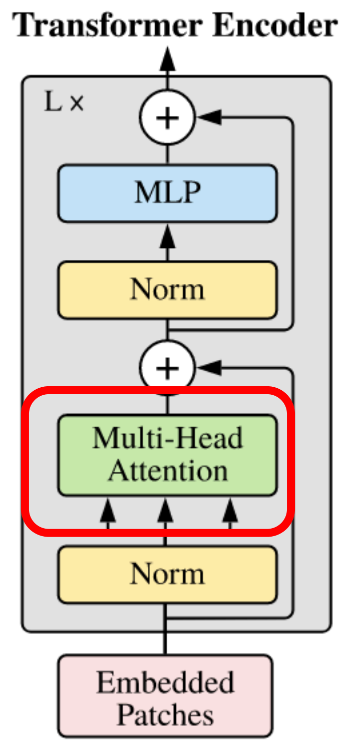
$$V = XW_v$$

2. Compute the attention matrix that measures the inter-tokens similarity

$$A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$$

3. Ponderate the **V**alue matrix with the attention matrix

$$Z = AV$$



[Dosovitskiy, Beyer, Kolesnikov et al. ICLR 2021]

Multi-heads Self-Attention

$$Q_1 = XW_{q_1} \text{ and } Q_2 = XW_{q_2}$$

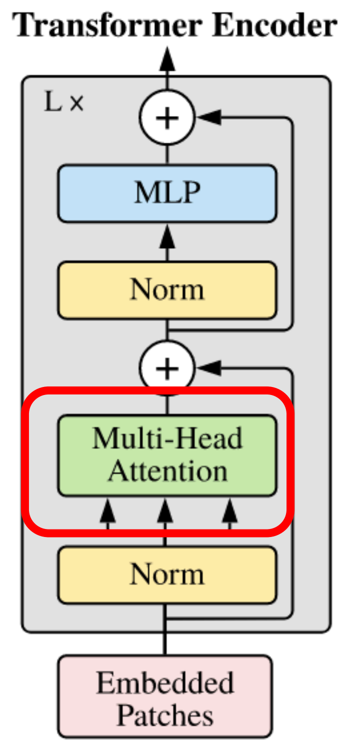
$$K_1 = XW_{k_1} \text{ and } K_2 = XW_{k_2}$$

$$V_1 = XW_{v_1} \text{ and } V_2 = XW_{v_2}$$

$$Z_1 = \text{softmax}(\frac{Q_1 K_1^T}{\sqrt{d}}) \text{ and } Z_2 = \text{softmax}(\frac{Q_2 K_2^T}{\sqrt{d}})$$

$$Z' = [Z_1 Z_2] \in \mathbb{R}^{T \times 2d}$$

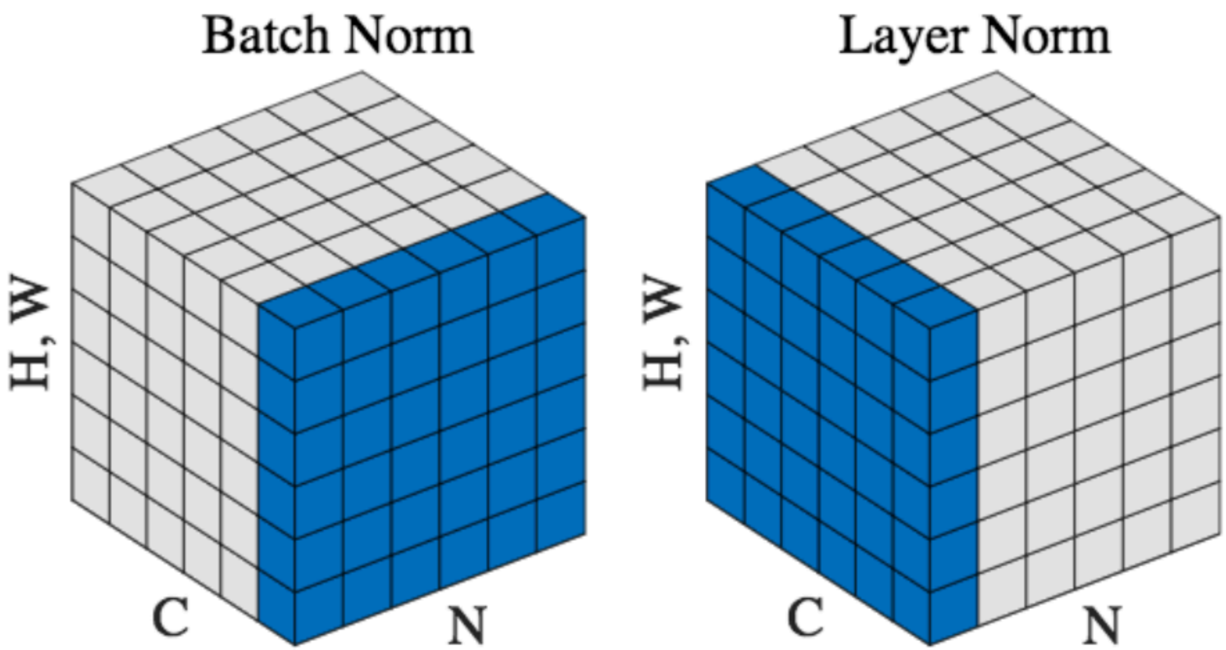
$$Z = Z' W_o$$

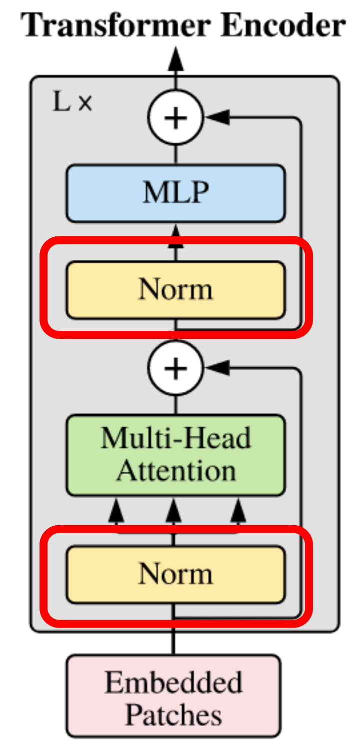


In practice, the self-attention is done multiple times in parallel, with different **heads**.

[Dosovitskiy, Beyer, Kolesnikov et al. ICLR 2021]

Layer Norm





- Normalize per channel
- Do not track running mean/std → same behavior between train and test!
- Used at first for RNN in NLP

[Ba et al. NeurIPS Workshop 2016], image source

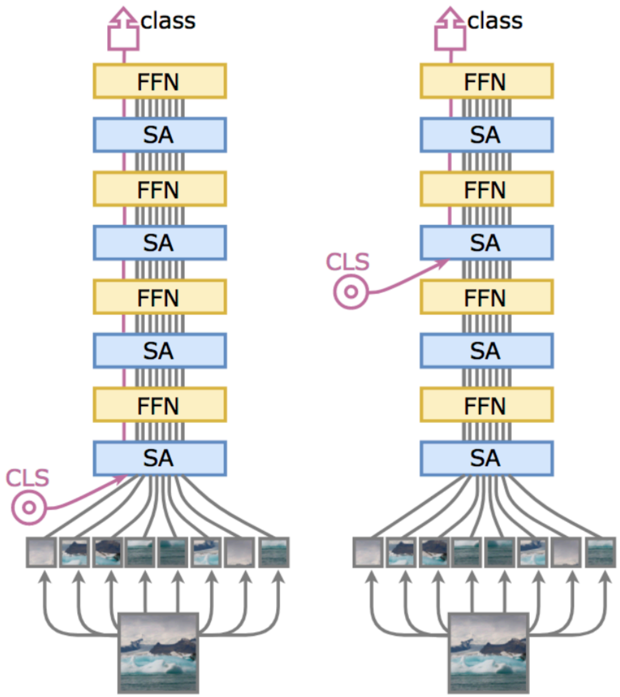
DeiT

												top-1 accuracy	
Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	pre-trained 224^2	fine-tuned 384^2
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8 ±0.2	83.1 ±0.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3
	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

- Transformers are hard to train because they lack **inductive bias**, and thus needs way more data than a ConvNet
- DeiT partially close this gap by using tons of **data augmentations** and **regularizations**

[Touvron et al. ICML 2021]

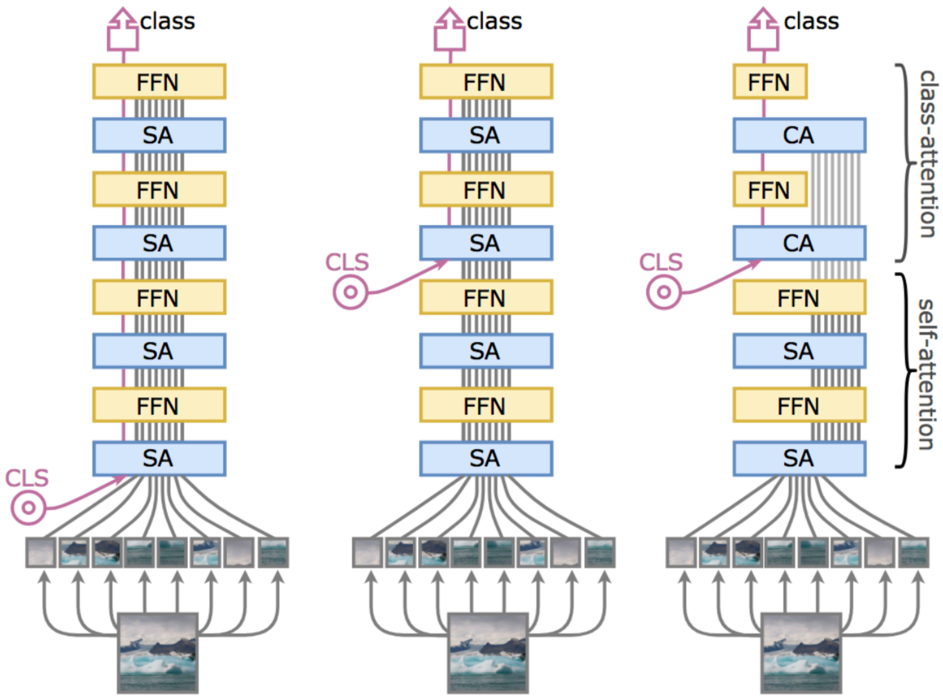
CaiT



- Inserting the class tokens in later blocks may prove beneficial

[Touvron et al. ICCV 2021]

CaiT



$$z = [x_{\text{class}}, x_{\text{patches}}]$$

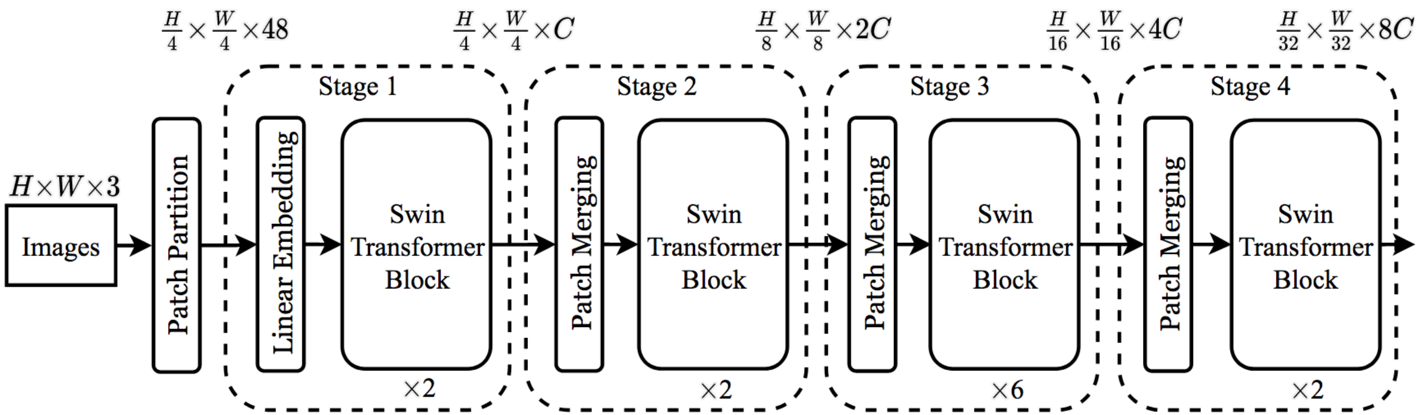
$$Q = W_q\, x_{\text{class}} + b_q,$$
$$K = W_k\, z + b_k,$$
$$V = W_v\, z + b_v.$$

$$A = \text{Softmax}(Q.K^T / \sqrt{d/h})$$

- Inserting the class tokens in later blocks may prove beneficial
- Add Class-Attention → linear complexity w.r.t the number of patches!

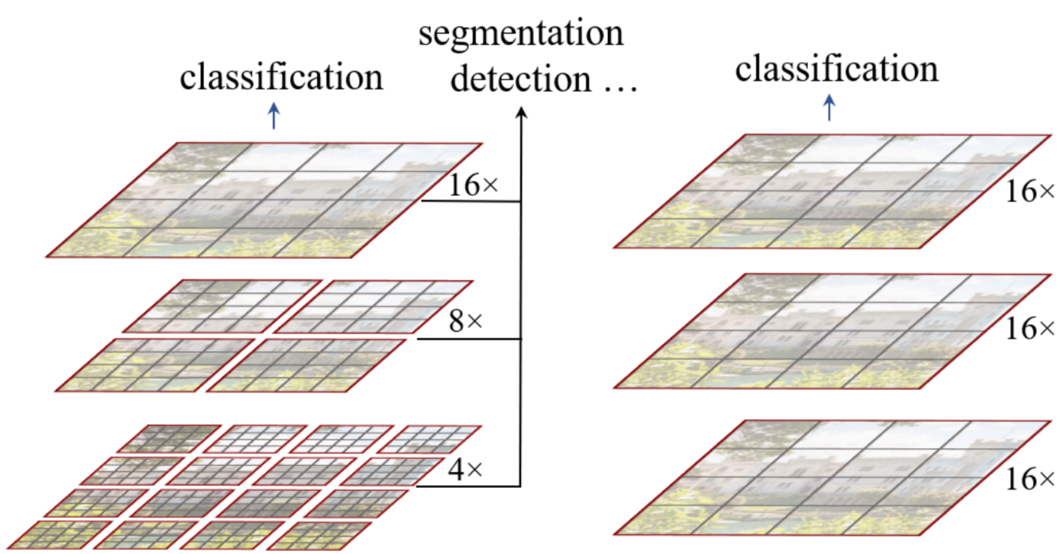
[Touvron et al. ICCV 2021]

Swin Transformer



- Compute self-attention only on local area, reducing computational complexity

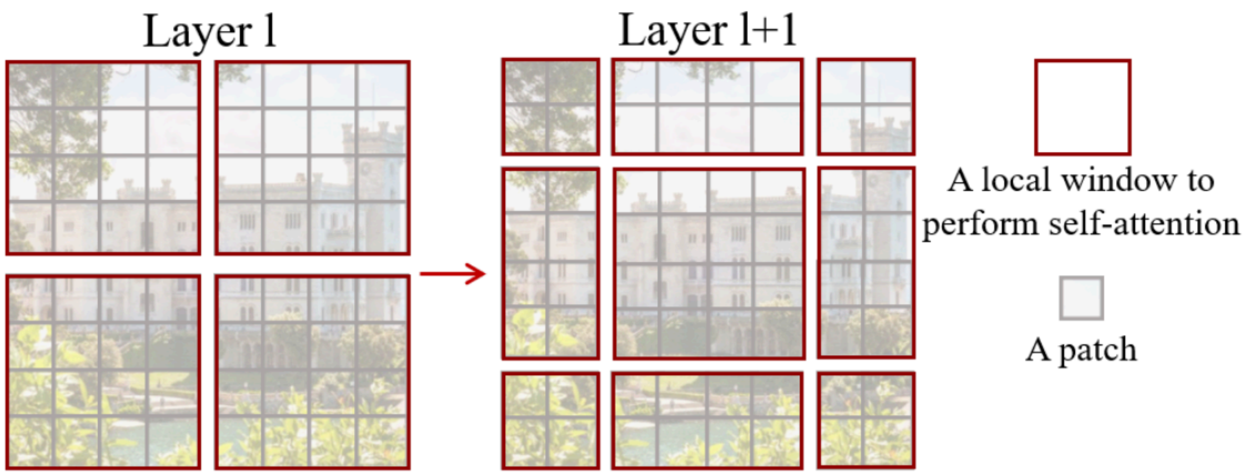
- Token merging is a concatenation alongside the embed dimension axis



(a) Swin Transformer (ours) (b) ViT

[Liu et al. ICCV 2021]

Swin Transformer

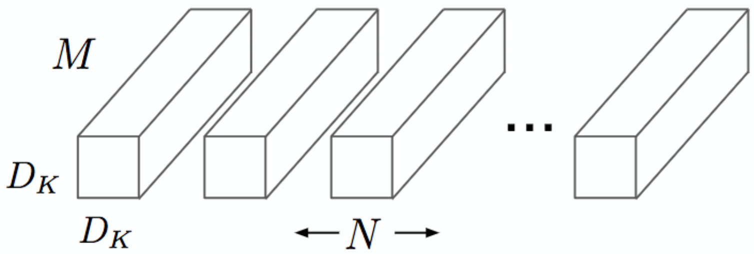


- Each layer shift the windows (top-left to bottom-right)
- Allow overlap and thus communication between areas of previous layer

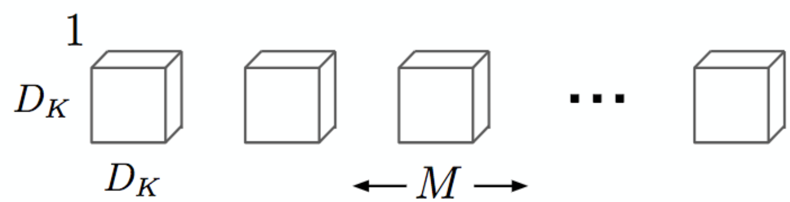
[Liu et al. ICCV 2021]

MLP Comeback!

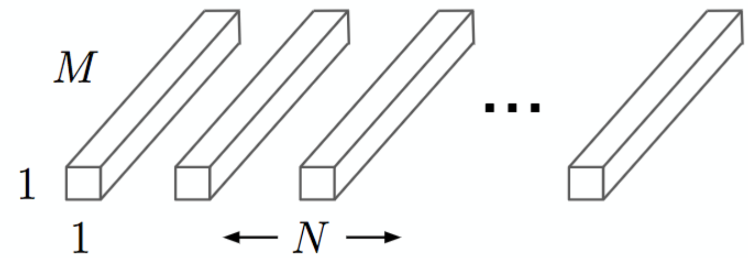
Separable Convolutions



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

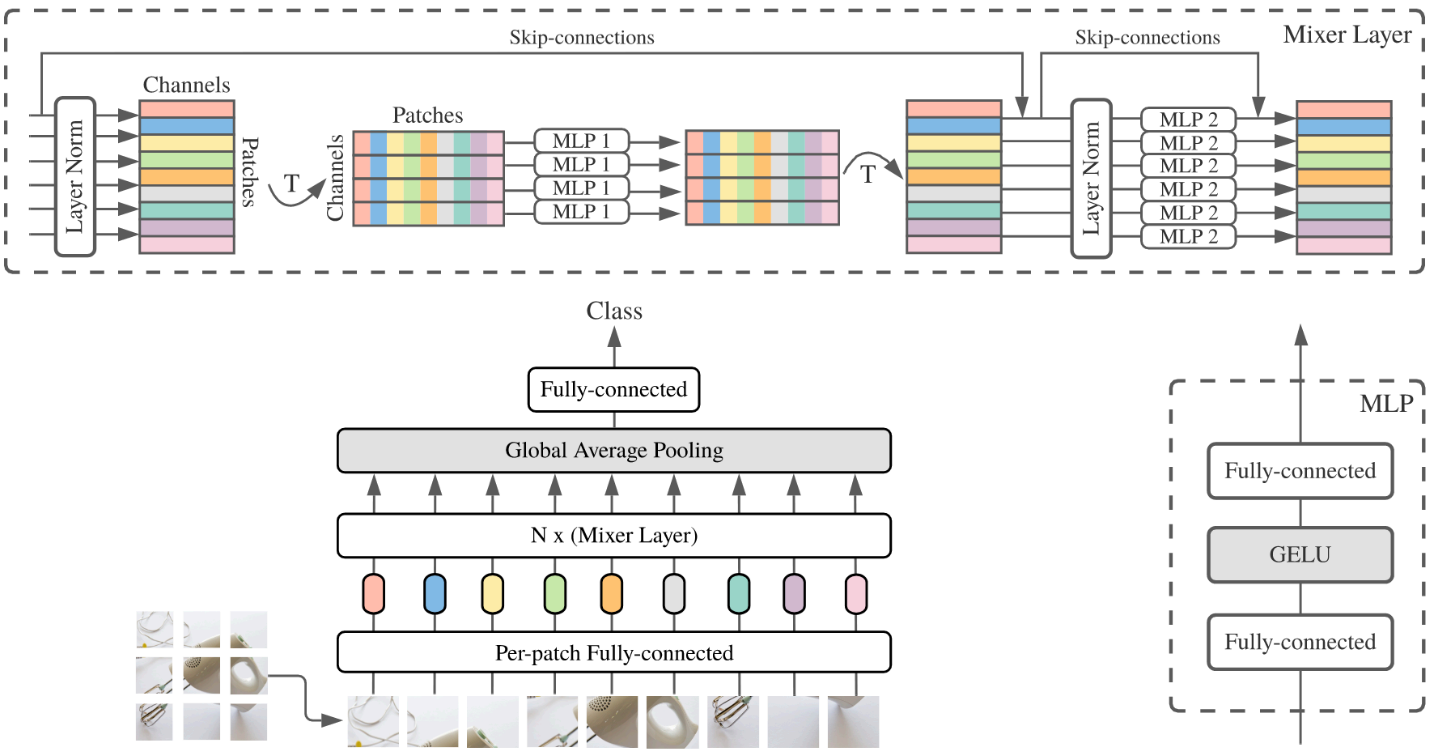


(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

- **Depthwise convolutions**: doesn't mix input channels
- **Pointwise convolutions**: doesn't mix spatial dimensions

[Chollet, arXiv 2016]

MLP-Mixer



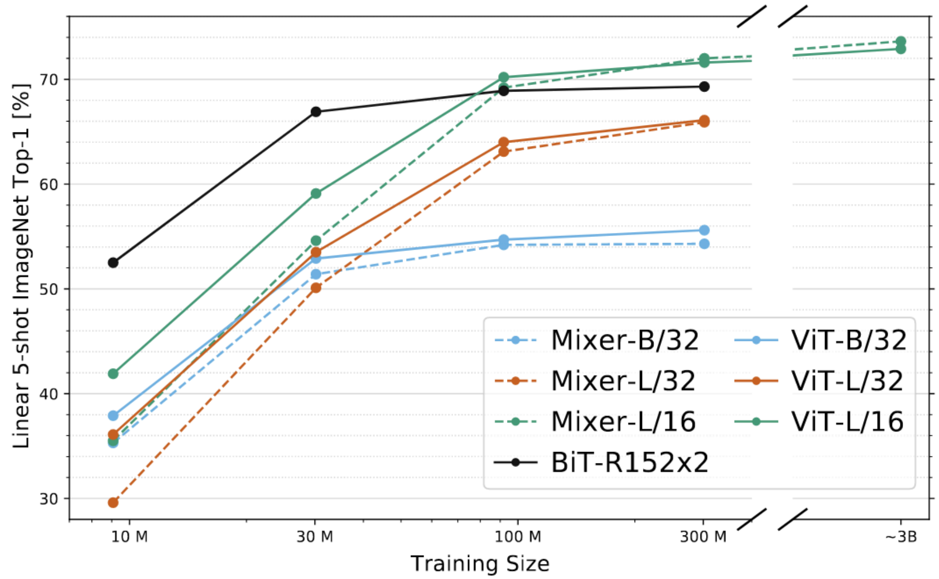
- Similarly to Separable Convolutions, apply a MLP on the channels dimension and a MLP on the patches dimensions

- Multiple other papers had the same idea at the same time (including ResMLP)

[Tolstikhin and Houlsby and Kolesnikov and Beyer et al. arXiv 2021]

MLP-Mixer

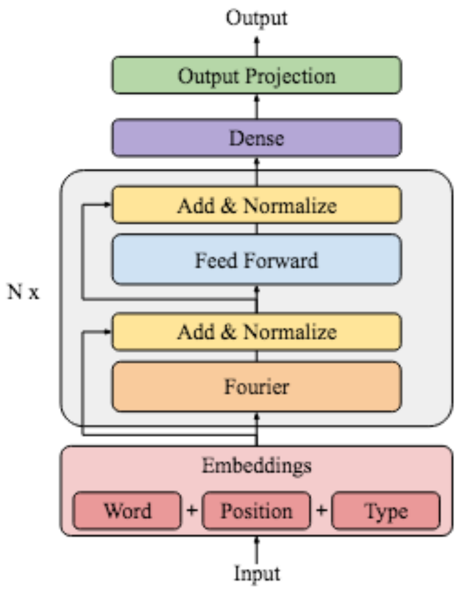
	ImNet top-1	ReaL top-1	Avg 5 top-1	VTAB-1k 19 tasks	Throughput img/sec/core	TPUv3 core-days
Pre-trained on ImageNet-21k (public)						
• HaloNet [51]	85.8	—	—	—	120	0.10k
• Mixer-L/16	84.15	87.86	93.91	74.95	105	0.41k
• ViT-L/16 [14]	85.30	88.62	94.39	72.72	32	0.18k
• BiT-R152x4 [22]	85.39	—	94.04	70.64	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



- However needs even more data than transformers based (ViT)
- Convolutions models (here BiT-R152x2) are still much more data efficient when training from scratch
    - However, some advocates that most future applications will be based on those new large (transformer, mlp, etc.) models pretrained on large-scale data
    - See [Bommasani et al. arXiv 2021]

Transformer & MLP-Mixer relation

Fourier Transformer



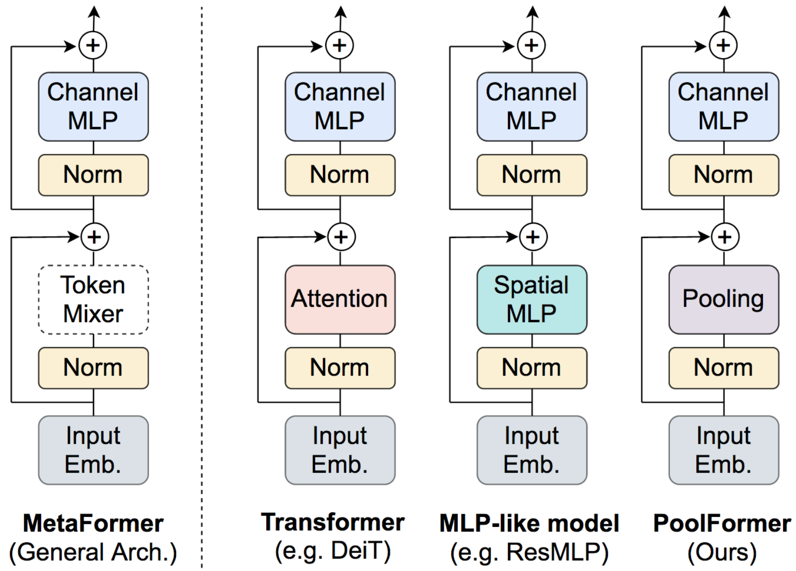
Fourier transform that can be expressed as a matrix multiplication with this constant matrix:

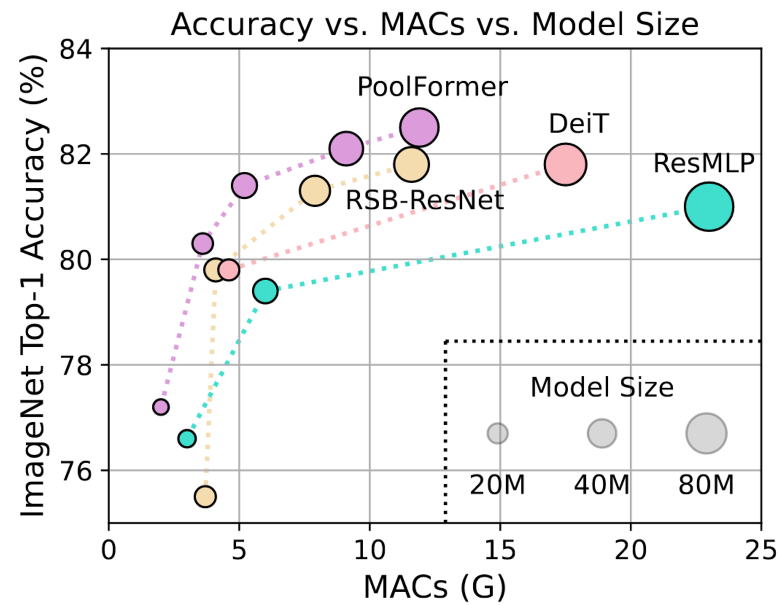
$$W_{nk} = \left(e^{-\frac{2\pi i}{N}nk}/\sqrt{N}\right)$$

- Not as good as Self-Attention, but still impressive results

- Means that *"Attention is **NOT** all you need"*, but rather a way to combine inputs
    - Likewise convolutions combine pixels through the increasing receptive field

Meta-Former





- Even simple pooling as token mixer seems to reach SotA results
- For all training config (optimizer, scheduling, regularizations, etc.) is important

- PoolFormer was released 22 Nov 2021, so everything is very new

→ Still a very active research problem
→ Conclusions may improve back all fields (Vision, NLP, Speech, etc.)
→ Lot of recent work on multi-modal, e.g. combining Vision and NLP

[Yu et al. arXiv 2021]

Tricks that work for most architectures

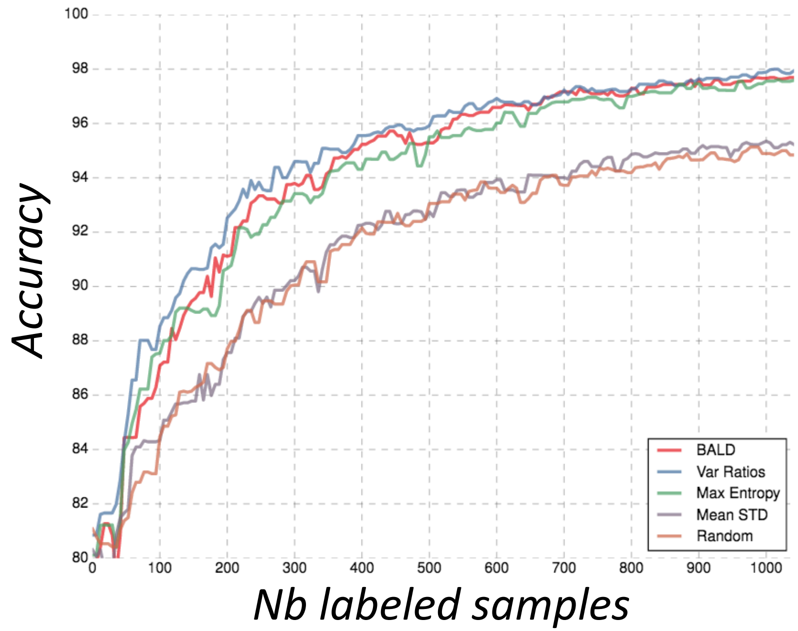
Train / Val / Test

- It’s extremely important to tune the hyperparameters on a val set

- In real-life, you often don’t have access to the full test set
    - And this test set may change constantly

- It’s also important to ensure that your train and val sets have the same distribution than the test set
    - Beware of the **sampling bias**, e.g. I’m only labeling images of cars oriented towards the front, but in the test / real-life I may see cars in other orientations
    - See [Torralba and Efros, CVPR 2011]

Active Learning



- More data is (almost) always better, see the Sutton's [Bitter Lesson]

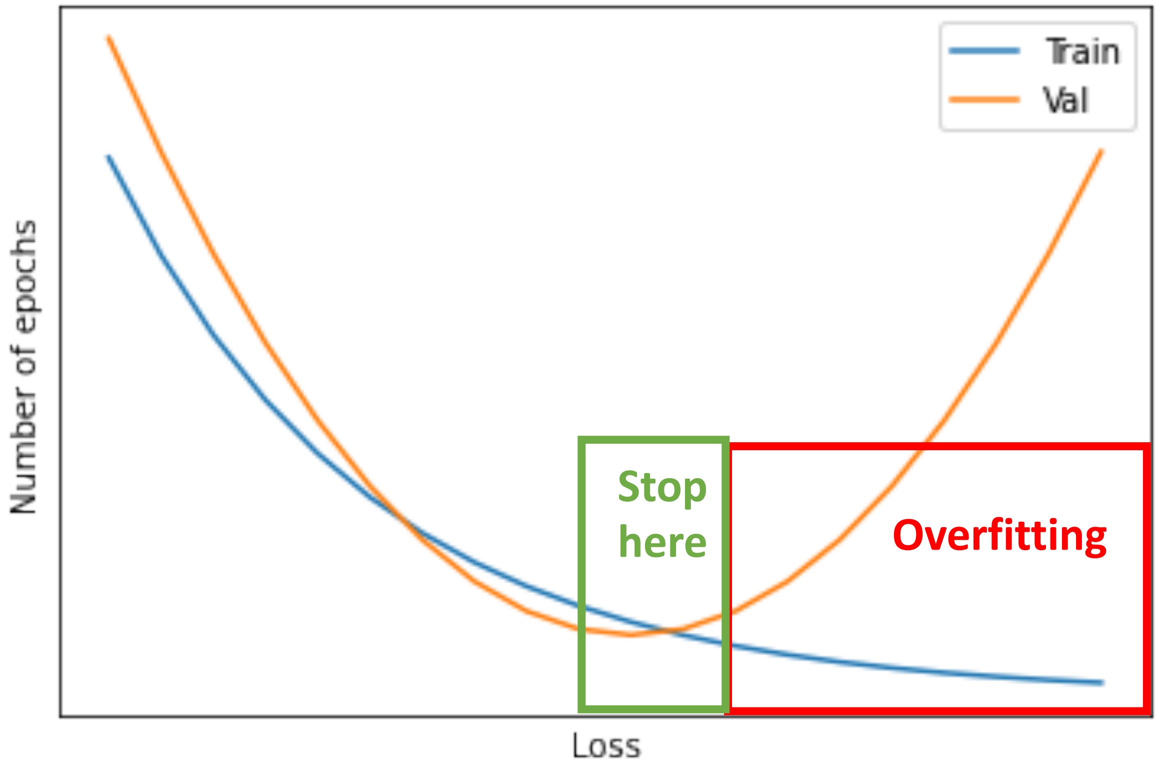
- But hand-labeling data is very costly, both in $ and time

- Active learning aims to determine which data to labelize in priority to be added to the training set
    - A lot of the literature is based on Bayesian stats, with Yarin Gal's team
    - But often done on small-scale datasets (MNIST, CIFAR)
    - And a random sampling is often quite competitive despite its simplicity

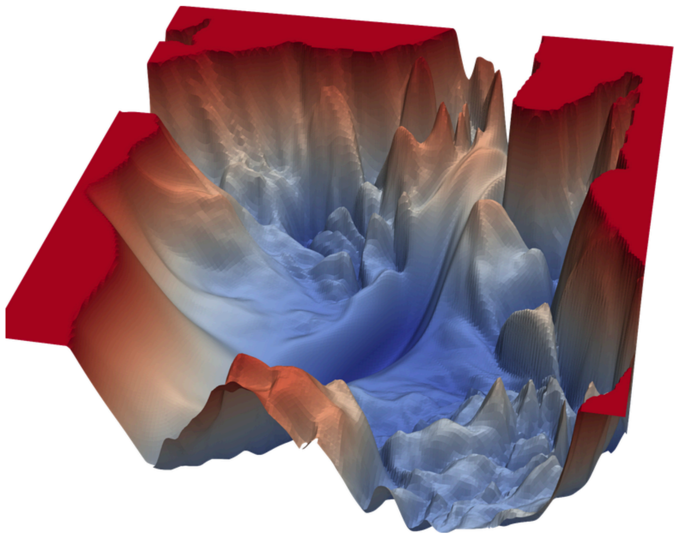
[Gal et al. ICML 2017]

Early Stopping

- Training for too long, and you start to overfit. So when to stop?
- Monitor a metric (accuracy, loss, etc.) on the **VALIDATION SET** (!) and if this metric gets worse for X epochs, stop training
- ”*A beautiful free lunch*” according to Turing award’s Geoffrey Hinton



Decrease Learning Rate



- A high learning rate during the beginning of the training may help
  - Acts as a regularization by **skipping the local minima that are too sharp** and thus usually generalize less

- Then **decrease learning rate gradually to go deeper in a local minim**a towards the training end
  - Either decrease learning rate (usually divided by 10) at particular epochs
  - Or decrease if validation metric (loss, acc, etc.) doesn't improve

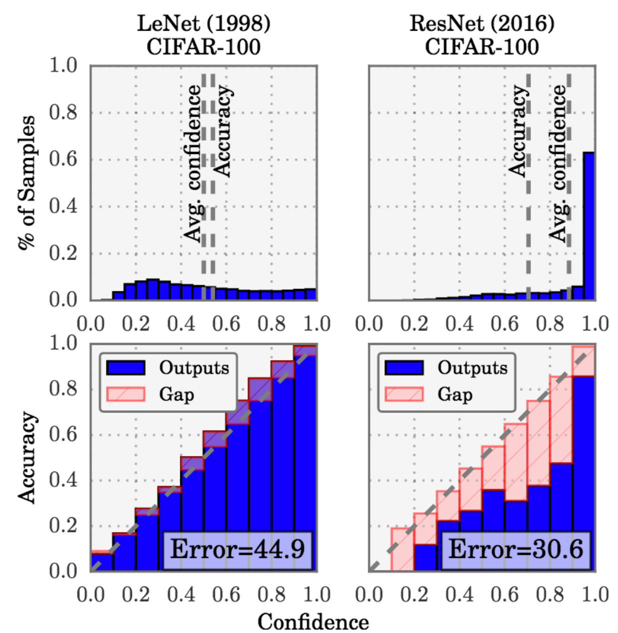
- Some scheduling as **Cosine** decreases and increases (a little less) repetitively

Smooth Labeling

$$y_k^{LS} = y_k(1-\alpha) + \alpha/K$$

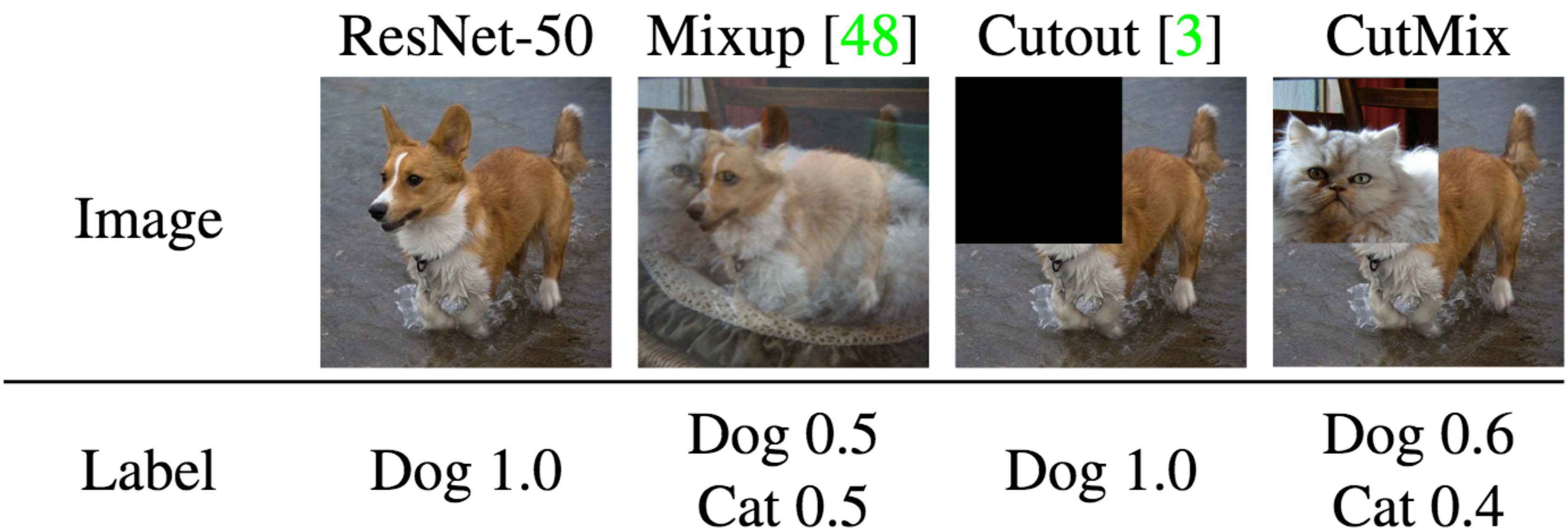
[0, 0, 1, 0] with a $\alpha = 0.1$ produces [0.025, 0.025, 1, 0.025]

- Avoid overconfidence in the model, when confidence is always close to 0.999…
  - And thus reduce overfitting
- Avoid miscalibration where model confidence is not correlated to model accuracy



Smooth labeling: [Muller et al. NeurIPS 2019], Miscalibration: [Guo et al. ICML 2017]

MixUp & Co



- Mix two images and their labels together
  - In practice MixUp mix them with factors like 0.9/0.1 (not 0.5/0.5 as on the image)

- Acts as regularization to reduce overfitting

- A LOT of alternative to MixUp exists (CutOut, CutMix, FixMatch, MixMo, PuzzleMix, etc.)

Mixup: [Zhang et al. ICLR 2017]

Knowledge Distillation

- Train one super-mega-large model (called teacher)
- **Distill** the knowledge of the teacher onto a smaller model (called student)

- In practice, train students as usual but add a another loss:
  - **KL-divergence** between the probabilities of the teacher and the student
  - The probabilities acts as **dark knowledge** with extra information
    - (aka if the teacher say this is a dog with 0.7 confidence, we know it's a dog, but it's probably not the most archetypal dog ever)
  - Often add a **temperature** $\boldsymbol{T}$ on the logits before softmax
    - If $T > 1$, reduces the sharpness of the probabilities leading too more useful info

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

[Hinton et al. NeurIPS Workshop 2014]

Thresholding & Open-Set

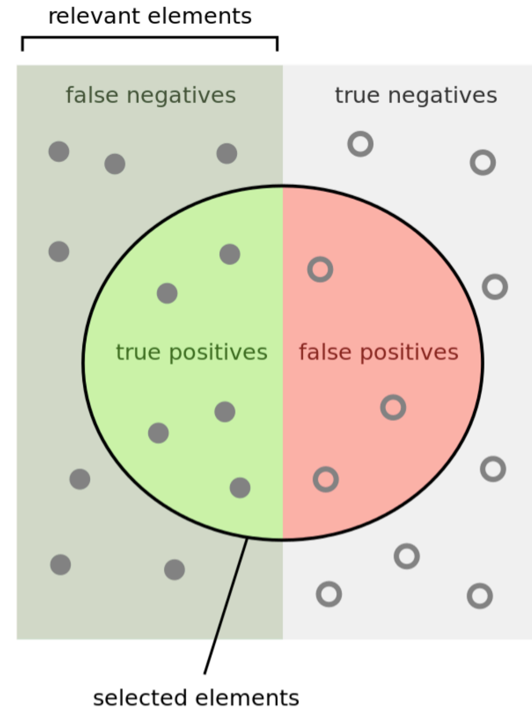
- In the real-life test set, you don't always want to predict on all images

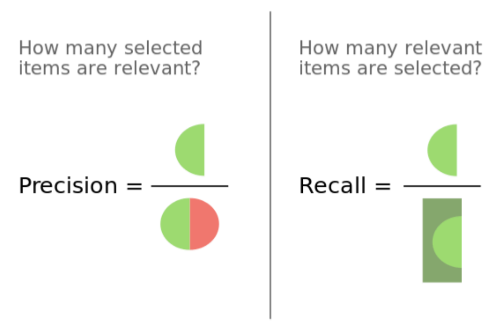
- Example:
  - *at Heuritech we predict fashion trends from images scrapped from Instagram. Given an image of dog, my model should predict any trend!*

- You choose to discard model prediction if its confidence is lower than a threshold
  - One threshold per class is better
  - Compute threshold on the validation set (which needs to have negative images!)

Accuracy vs Precision+Recall=F1

- In real-life classes are never equally balanced
- If there are 90% of *dogs*, and 10% of *cats*, and your model has less than 90% of accuracy it's bad…
  - Answering *dog* every times gives 90% accuracy
- Best to use other metrics like **Precision** and **Recall** or their combination the **F1-Score**
- **Recall** is particularly useful in **open-set** where your model shouldn't predict on all images





$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}$$

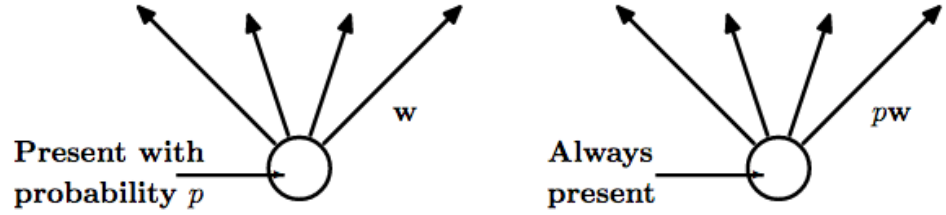
Wikipedia

Dropout

Randomly drop unit during a forward pass.

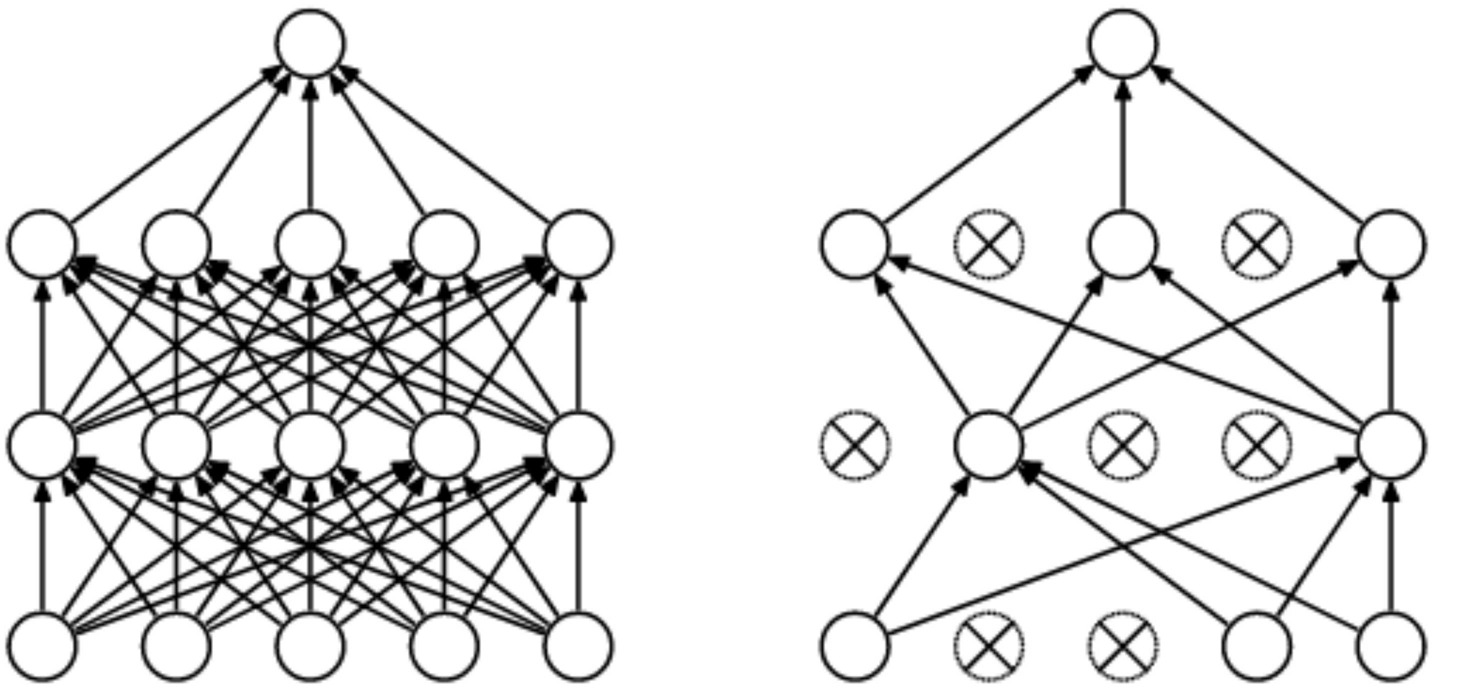
Drastically reduce overfitting:
- Sort of ensemble of networks
- Force all units to contribute

Usually only for fully connected layers.



(a) At training time

(b) At test time

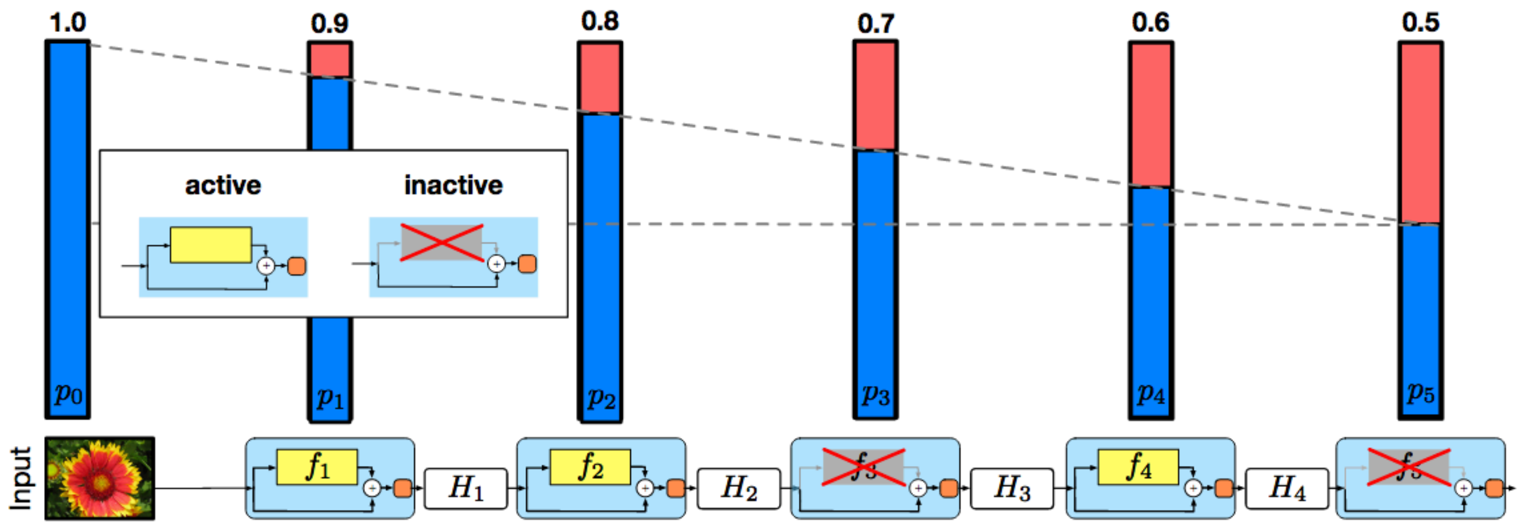


(a) Standard Neural Net

(b) After applying dropout.

[Sritastava et al. JMLR 2014]

Stochastic Depth



- Randomly drop whole residual block
- Only the identity shortcut is active

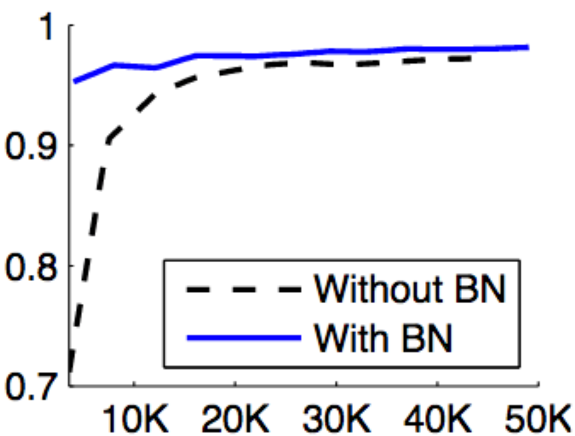
- Allow training even deeper networks, and is used in transformers

[Huang and Sun et al. NeurIPS 2016]

Batch Normalization

Normalize intermediary features.

During training with batch Statistics. During testing with running mean and std.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

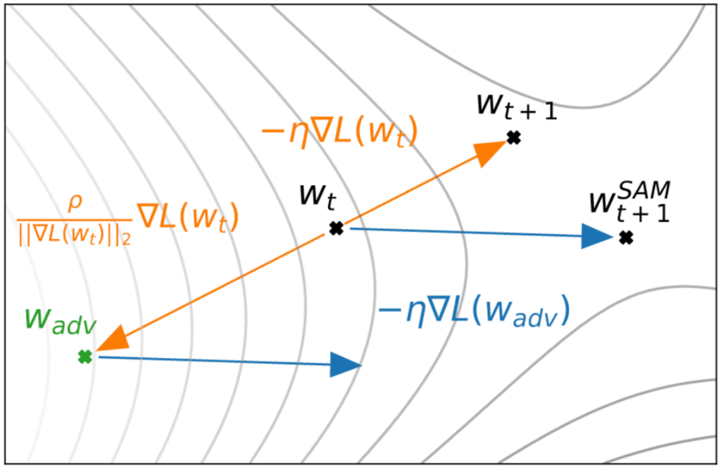
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

[Ioffe and Szegedy ICML 2015]

SAM: Sharpness-Aware Minimization



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 $\hat{\epsilon}(\boldsymbol{w})$ per equation 2;
- Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\epsilon}(\boldsymbol{w})}$;
- Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\boldsymbol{g}$;
- $t = t + 1$;

end

return $\boldsymbol{w}_t$

- Do not optimize network on a particular point of the parameters space but rather a **region**
- All neighbors parameters must also be good, leading to wider optimum and thus better generalization
- Needs twice more forward/backward…

[Foret et al. ICLR 2021]

Small break,
then coding session!