Deep Learning

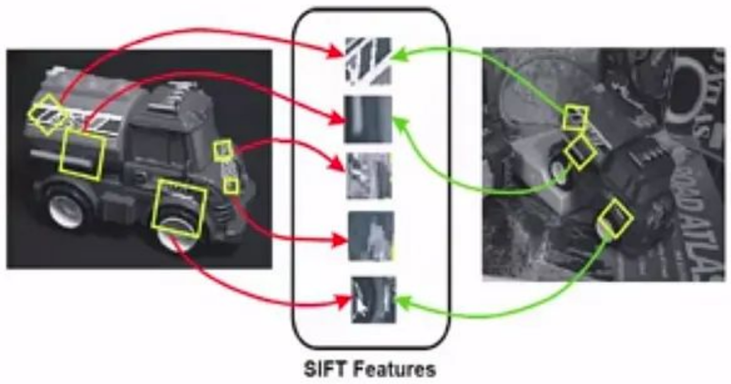
Lecture 6

Recap

- Global and local attention mechanisms
- Copy Mechanism
- Pointer Networks
- Image Captioning
- Transformer architecture
- Non-autoregressive Neural Machine Translation
- Semi-Autoregressive Neural Machine Translation

Classical approach: SIFT vectors

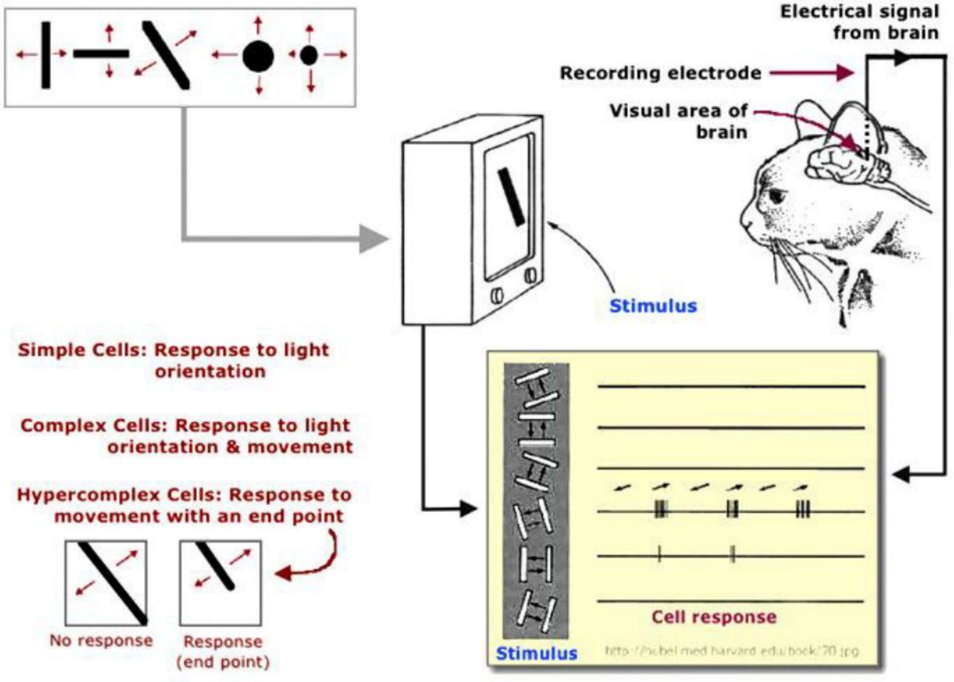
We should somehow extract features from images to build a classification model or match images.



Question: How can we do it?

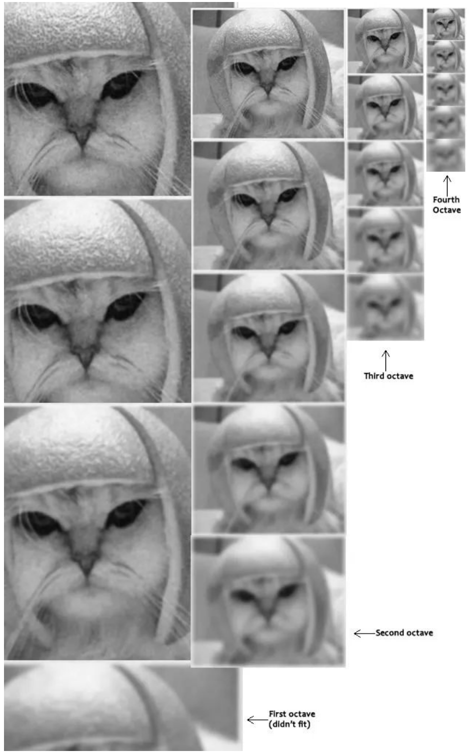
https://aishack.in/tutorials/sift-scale-invariant-feature-transform-scale-space/

Why edges are important?



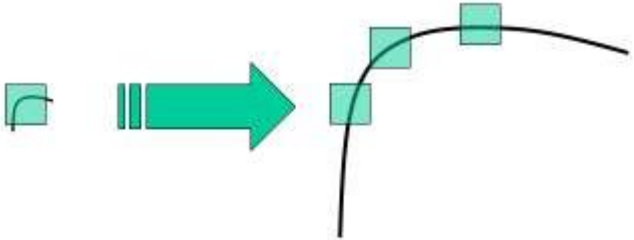
Hubel & Wiesel, 1959

SIFT vectors



Real world objects are meaningful only at a certain scale. You might see a sugar cube perfectly on a table. But if looking at the entire milky way, then it simply does not exist.

This multi-scale nature of objects is quite common in nature. And a scale space attempts to replicate this concept on digital images.



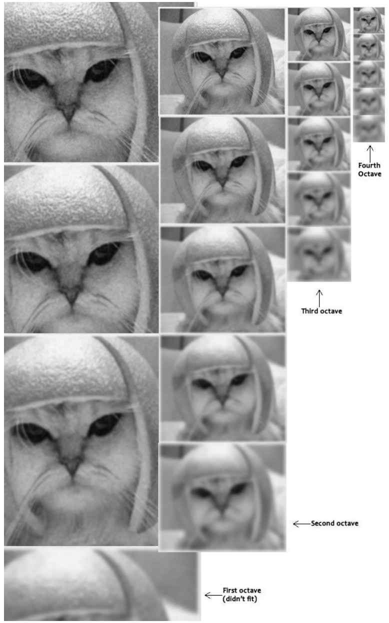
The basic concept of the image is edge, let's start by extracting edges

https://youtu.be/NfnWJUyUJYU?list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC&t=1220

Classical approach: SIFT vectors

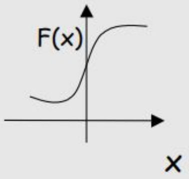
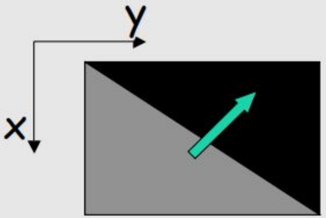
Using gaussian kernel to remove noise.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$



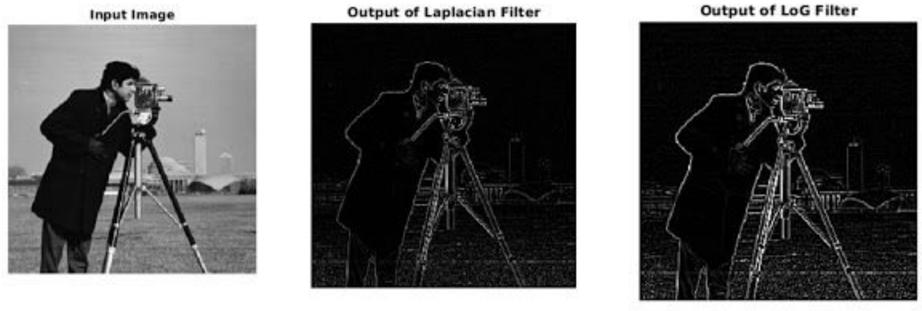
Then, repeat the operations on lower scale

Edge Detection Summary

	1D	2D
step edge	$I(x)$ 	$I(x,y)$ 
1st deriv	$\left\lvert \frac{dI(x)}{dx} \right\rvert > Th$	$\lvert\nabla I(x,y)\rvert = (I_x{}^2(x,y) + I_y{}^2(x,y))^{1/2} > Th$ $\tan\theta = I_x(x,y) / I_y(x,y)$
2nd deriv	$\frac{d^2I(x)}{dx^2} = 0$	$\nabla^2 I(x,y) = I_{xx}(x,y) + I_{yy}(x,y) = 0$ Laplacian

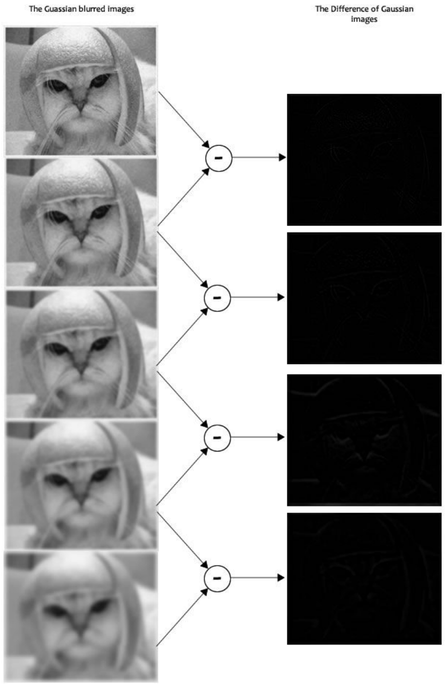
Laplacian of Gaussian (LoG)

Laplacian of Gaussian locates edges and corners on the image. These edges and corners are good for finding keypoints.

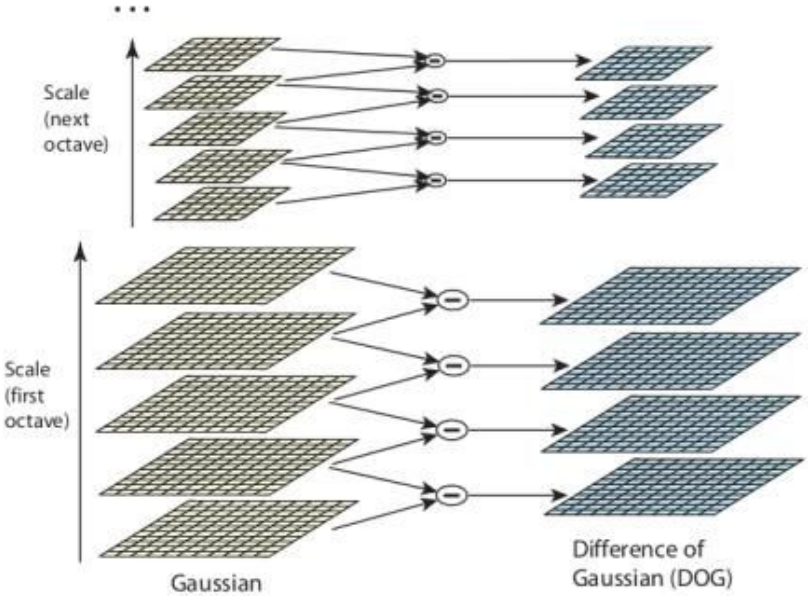


But it is computationally hard calculate. One can use simple approximation using **difference of Gaussians.**

LoG result



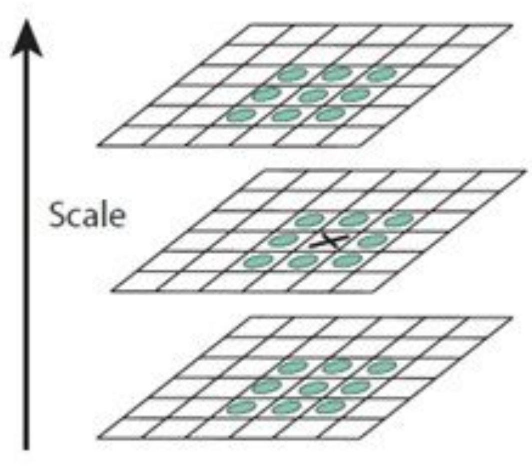
What we will get?

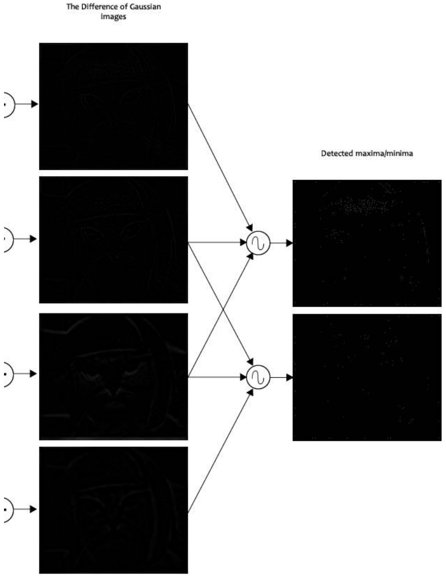


Repeating procedure over scales

Finding keypoints

Next, to find keypoints we are looking for maximum and minima. The author propose the following scheme: we compare the pixel marked with cross with all his 26 neighbours (see the left picture).





As we got 4 octaves (images) we can obtain 2 image with keypoints.

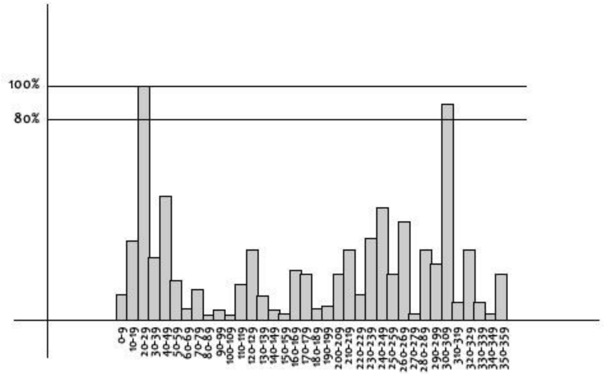
Orientation Assignment

The next thing is to assign an orientation to each keypoint. This orientation provides rotation invariance. The more invariance you have the better it is. Actually, it is very important when we do matching between different images. The same object can have different orientation. To calculate the orientation we use the following formulas:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

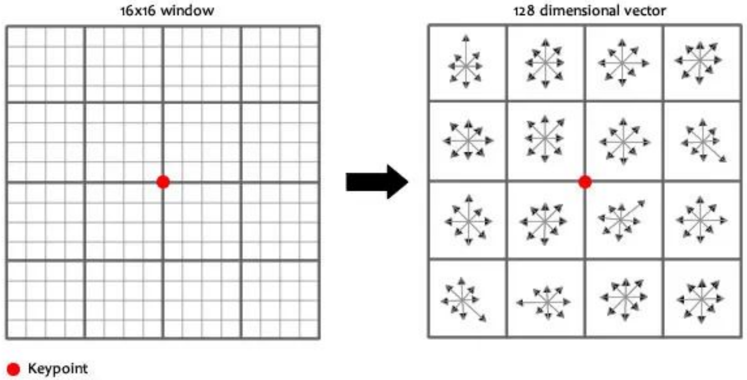
$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

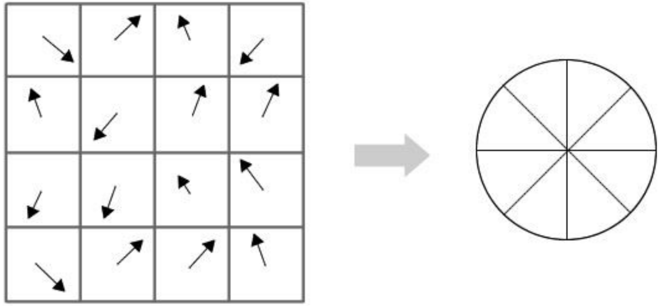
Then, we create a histogram with magnitude distribution. If distribution of current keypoint is more than 80% than we consider it as new keypoint



Keypoint descriptor

We want to generate a very unique fingerprint for the keypoint. It should be easy to calculate. We also want it to be relatively lenient when it is being compared against other keypoints. To do this, a 16x16 window around the keypoint. This 16x16 window is broken into sixteen 4x4 windows.

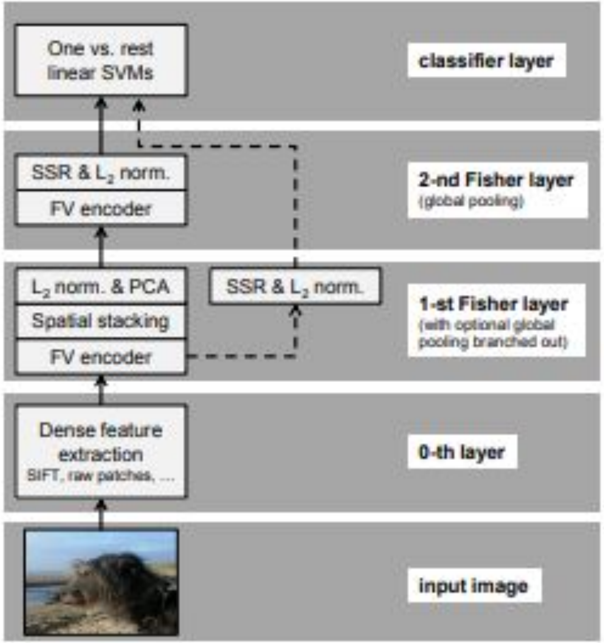


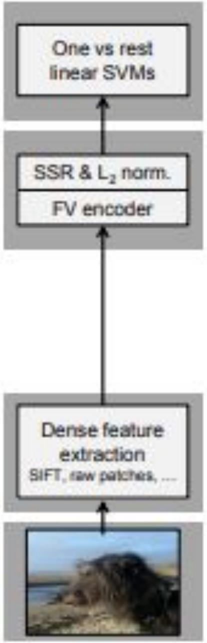


First, we create 16x16 window around the image

Then, inside a window we calculate distribution of gradients = 8 features

Classical solutions



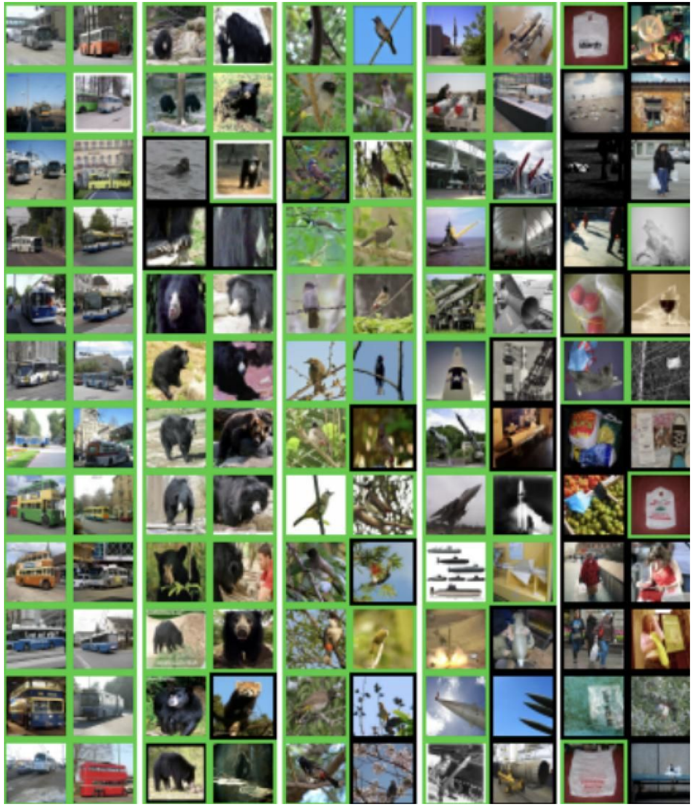


Before deep learning era solution for classification task has the following form:

- Manual feature extractor (e.g. SIFT)
- When some encoder to be able classify images of arbitrary size (Fisher vectors, codebooks)

Neural network solutions

ImageNet



"Everything in deep learning starts with data"

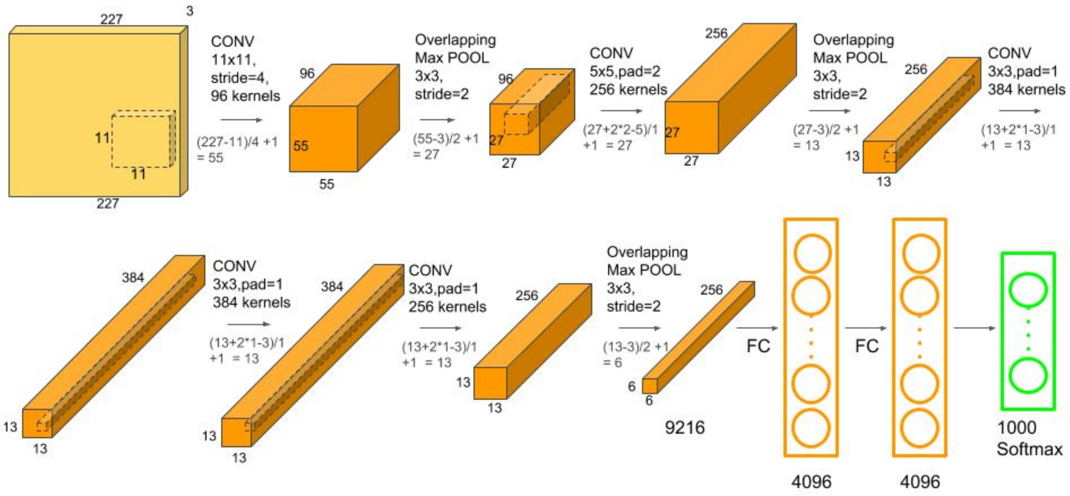
DL in Computer Vision started with ImageNet:

- ImageNet is a dataset of 1.2 M / 50k / 100k images (train/val/test)
- Consists of 1000 different image classes.

Datasets:

https://datasetsearch.research.google.com/
https://huggingface.co/docs/datasets/index

AlexNet



Architecture AlexNet

AlexNet: success factors

- Replaced tanh with ReLU (x6 speedup)
- Dropout + Augmentations
- 5 conv layers (11x11,5x5,3x3,3x3,3x3)
- Test-time augmentations (5 crops x horizontal flip)

Local normalization (analogue to batch normalization)

$$b^i_{x,y} = a^i_{x,y} \Bigg/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

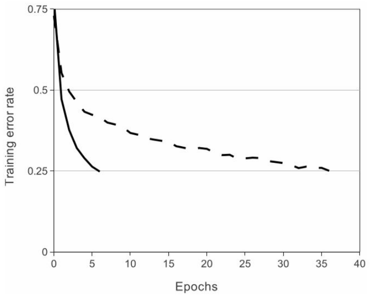
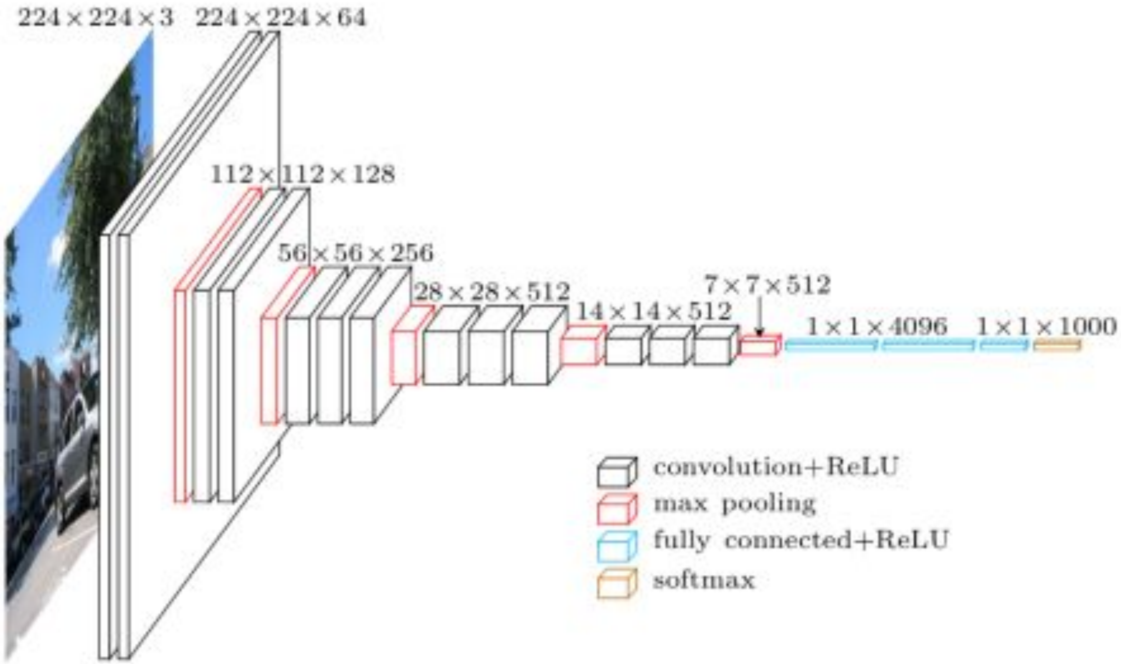
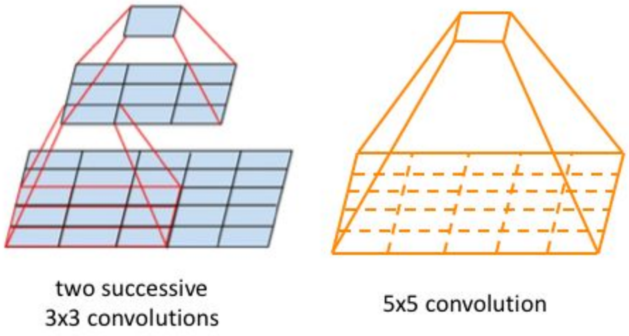


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

VGG


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

VGG: cascade of kernels



Instead of using, 5x5 conv authors proposed to use two 3x3 conv
It’s computationally more efficient (20 params vs 26 params), but the receptive field is equal

VGG: stagewise training

Initial architecture

The final architecture

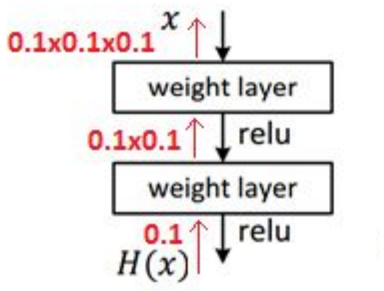
Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

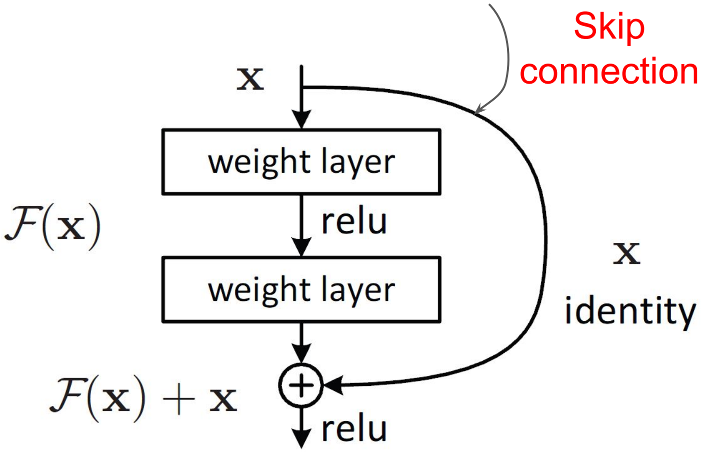
Instead of training all architecture end-to-end, they created a simple versions of architectures, trained them and added additional layers after the previous stage was completed.

Vanishing gradient problem

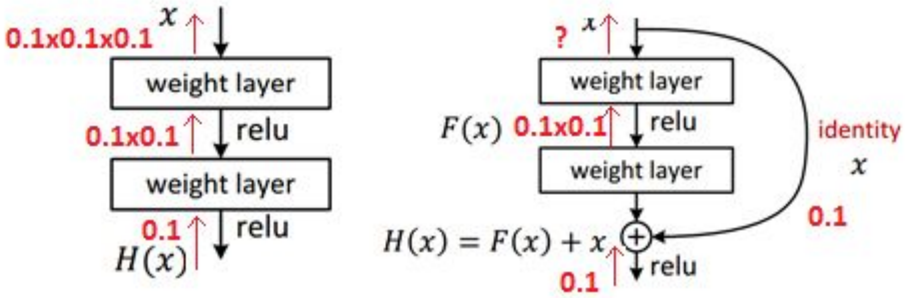
Let’s draw a little bit



Skip connection



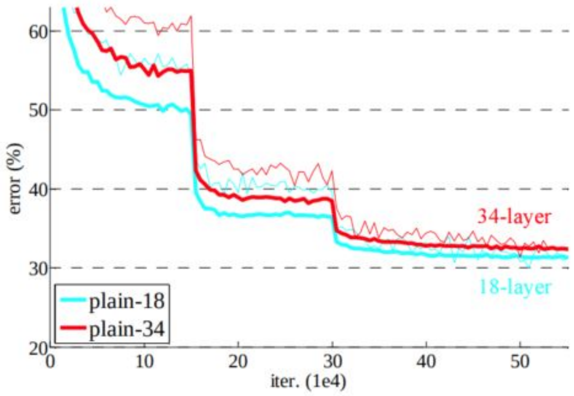
To prevent model from vanishing gradient problem one can use **skip connections**

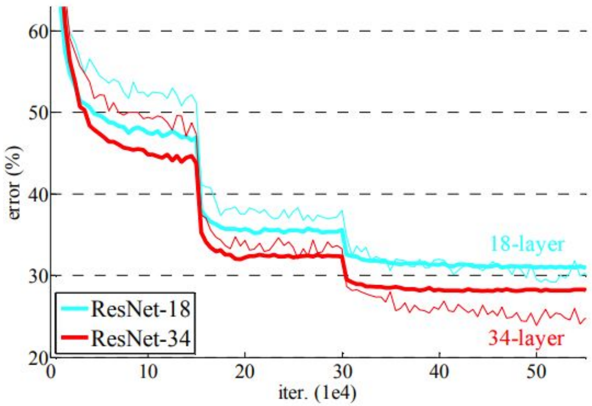


Creating highway allows to keep the gradient non-saturating + allows to pass information from bottom layer to top.

ResNet

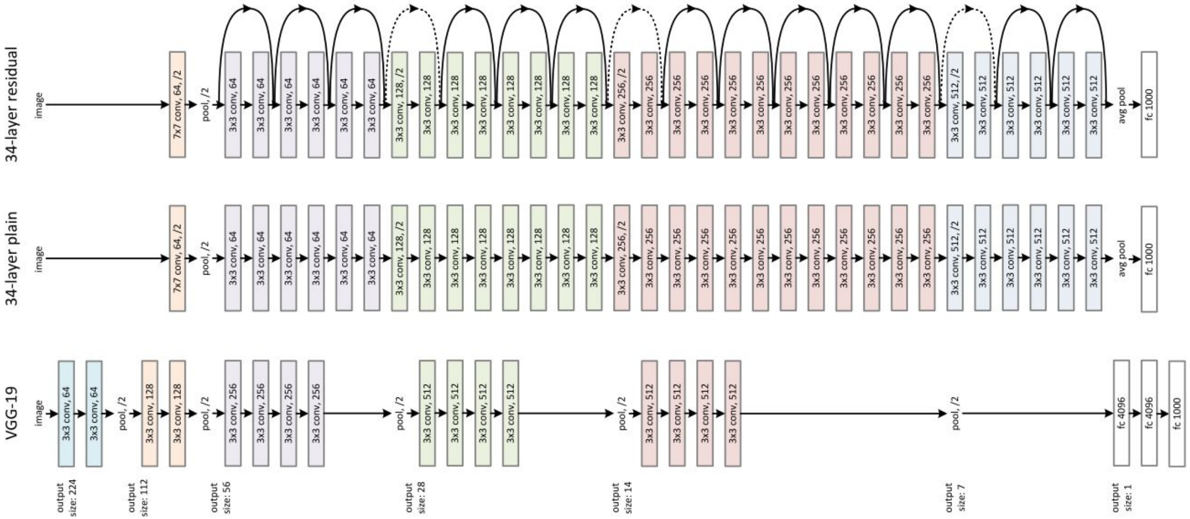
The result for base and deep model is the same.





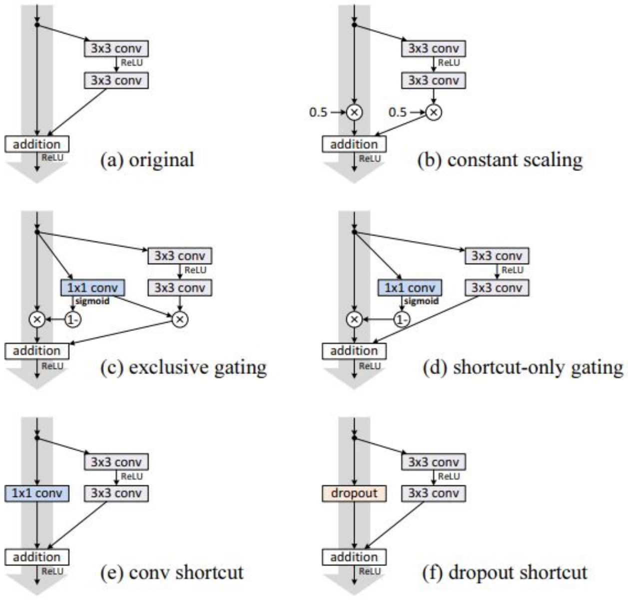
Results for the deeper model are better! Success!

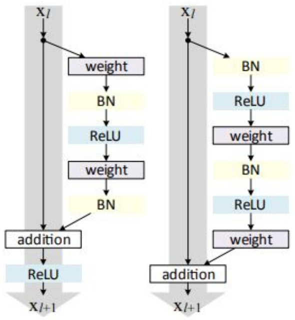
ResNet


34-layer residual
34-layer plain
VGG-19
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
fc 4096
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

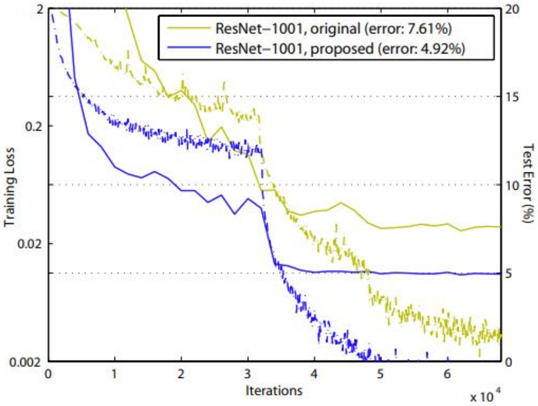
ResNet

What is the optimal residual layer form?





(a) original (b) proposed

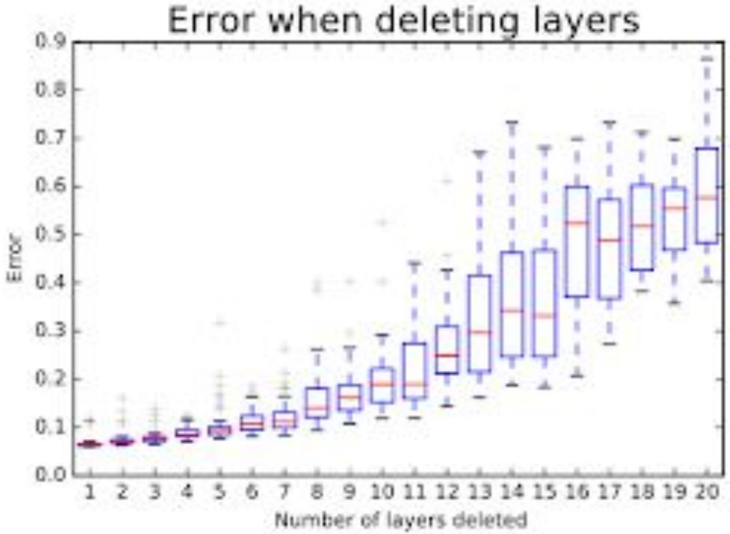


The best strategy for residual connection is $\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$

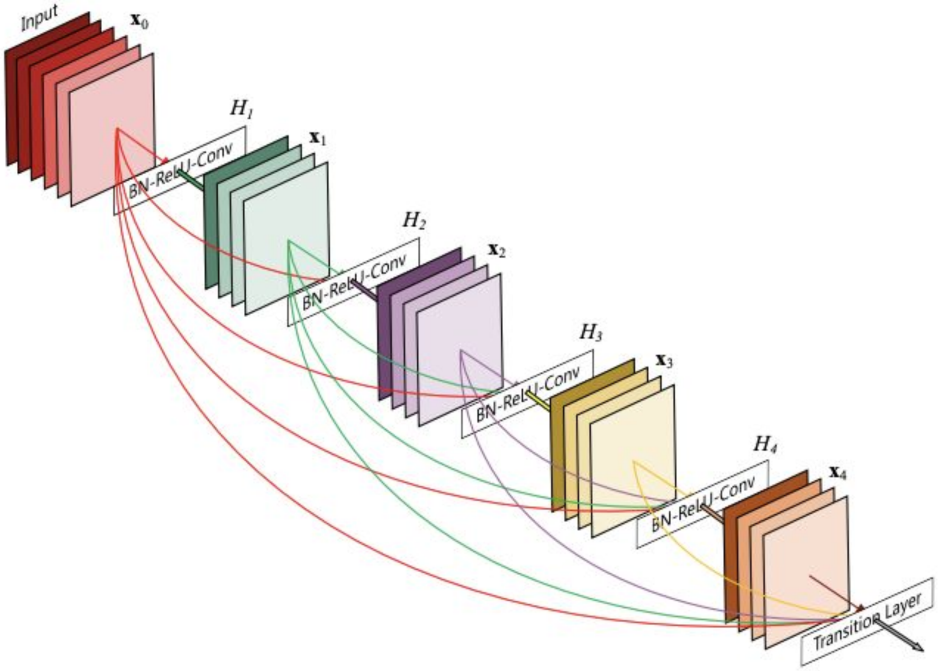
ResNet: cool feature

Interesting fact: ResNet can be a few layers out of ResNet and its performance wouldn't decrease too much.

This is achieved by the fact that ResNet has "workarounds" for information (skip-connection), and if one of them is interrupted. one of them, the information can still can still pass through the other



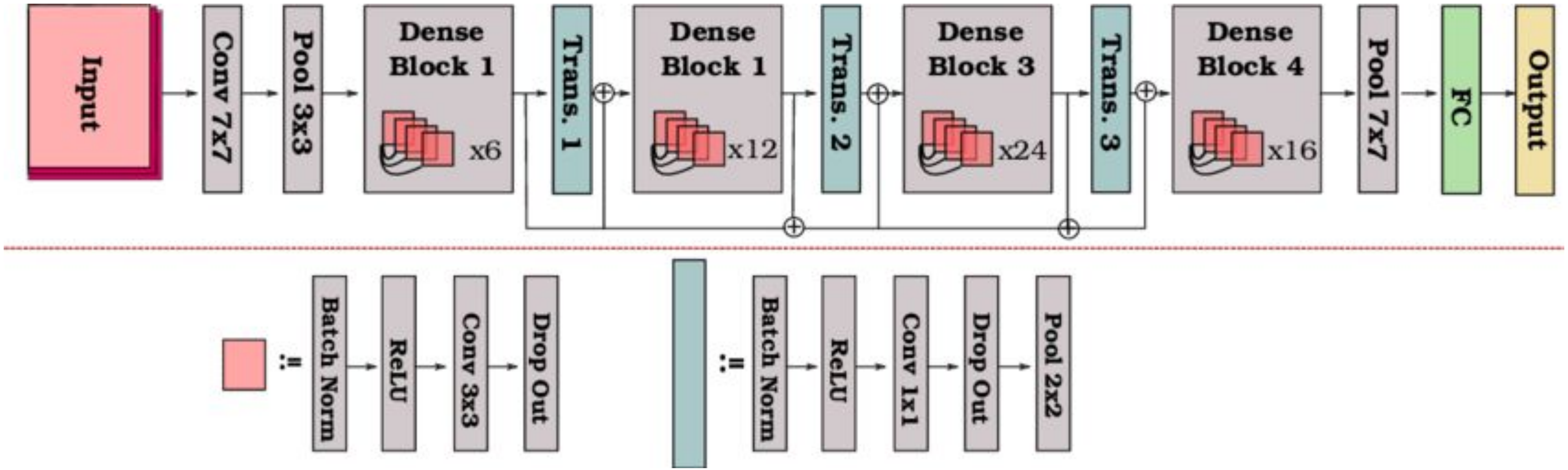
DenseNet



What if each successive layer of the network will receive as input all the outputs of all previous networks previous networks (instead of only one in ResNet)

DenseNet

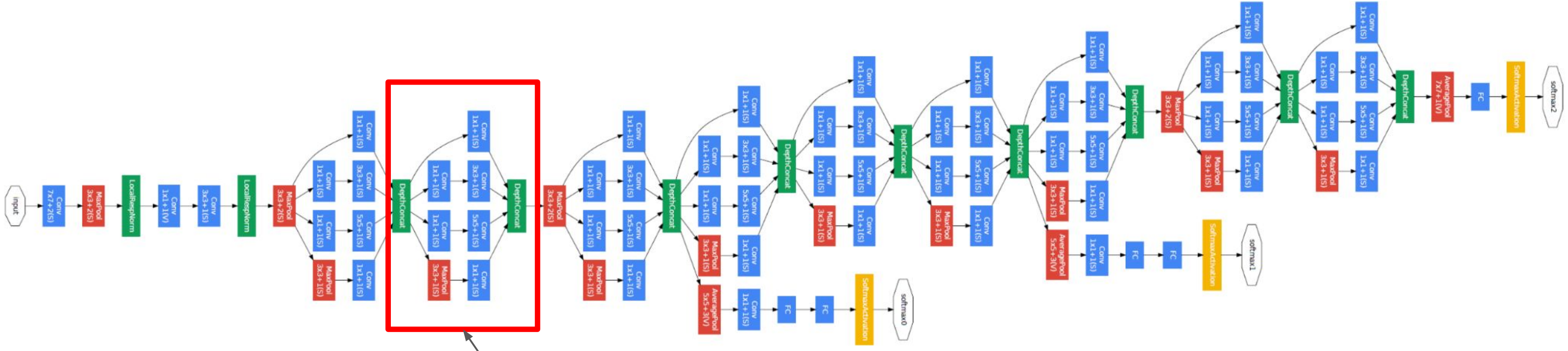
DenseNet


Input
Conv 7x7
Pool 3x3
Dense Block 1
x6
Trans. 1
Dense Block 1
x12
Trans. 2
Dense Block 3
x24
Trans. 3
Dense Block 4
x16
Pool 7x7
FC
Output
=
Batch Norm
ReLU
Conv 3x3
Drop Out
=
Batch Norm
ReLU
Conv 1x1
Drop Out
Pool 2x2

DenseNet

- strong gradient flow
- the number of layers and parameters is not very large
- conv layers emphasise a wider variety of features
- lower conv layers take into account low-complex patterns from higher layers, which can be useful for detecting some low-level patterns.

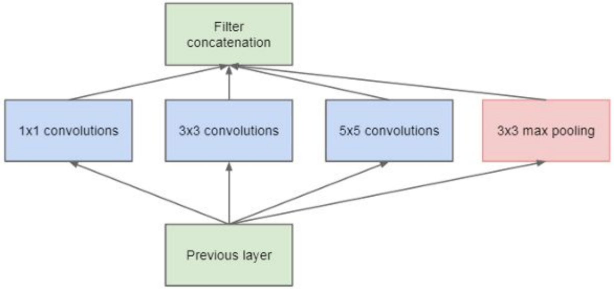
Inception | GoogLeNet



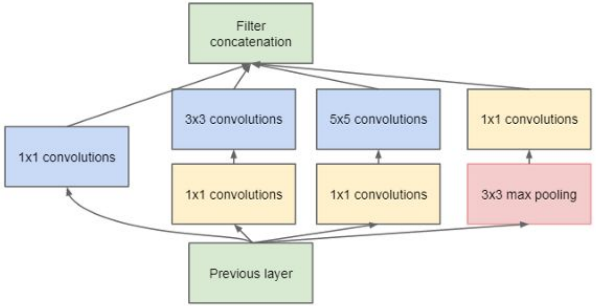
Inception module

- 22 layer
- Additional outputs for classification

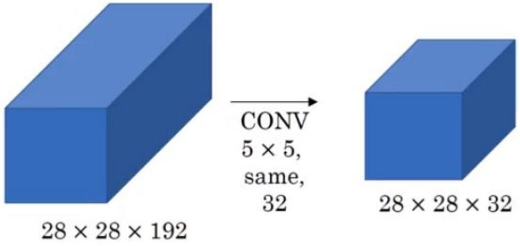
Inception module



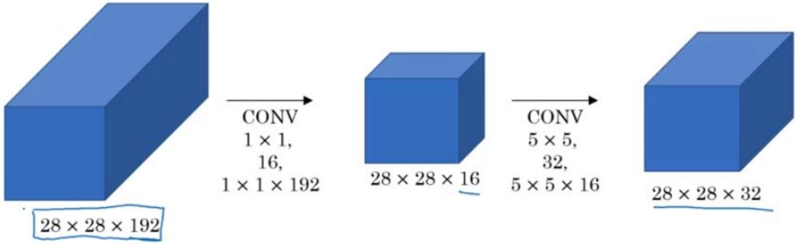
(a) Inception module, naïve version



(b) Inception module with dimension reductions



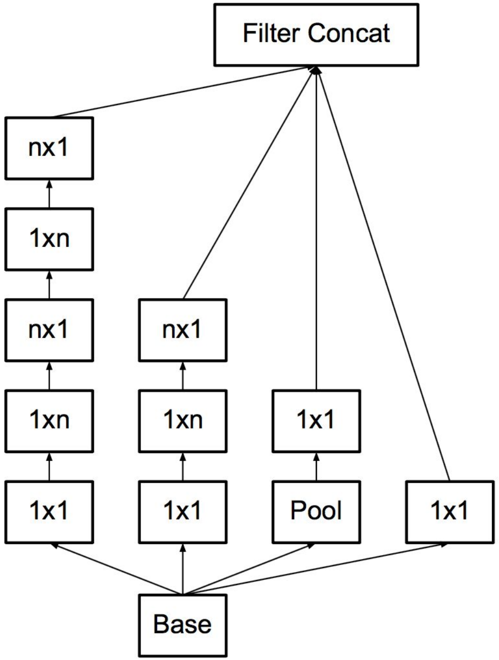
$\approx 120M$ calculations



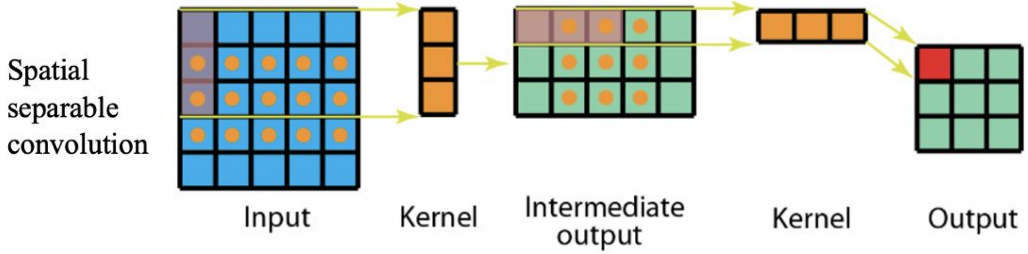
$\approx 12.4M$ calculations

ten times less calculations!

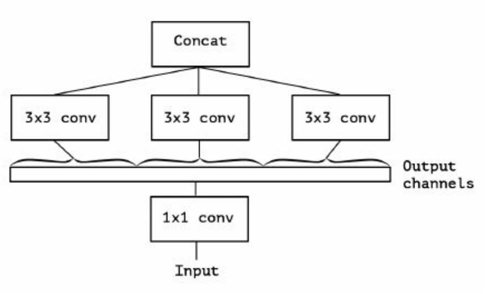
Inception v2, v3



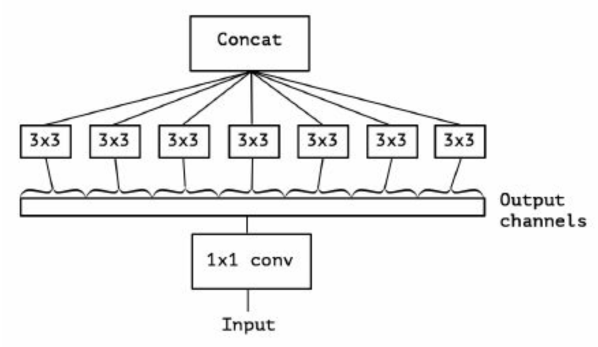
How to represent 3x3 convolution by composition of two one dimensional convolutions?



Xception



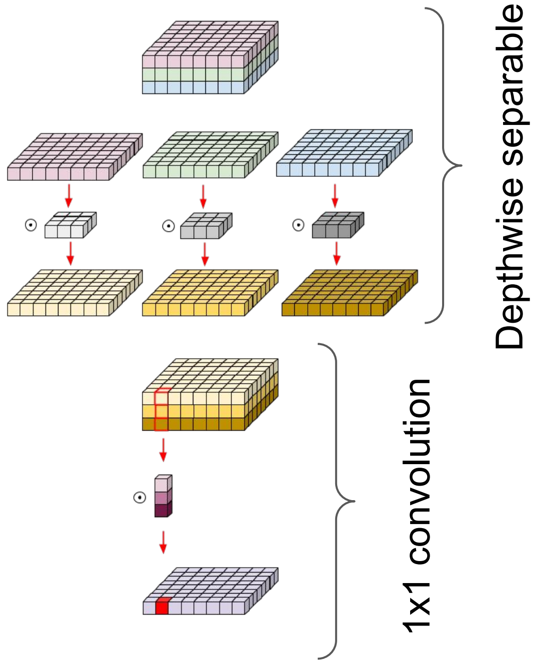
Standardized Inception module form



Xception module

Hypothesis: cross-channel correlations and spatial correlations are sufficiently decoupled that it is preferable not to map them jointly

Xception



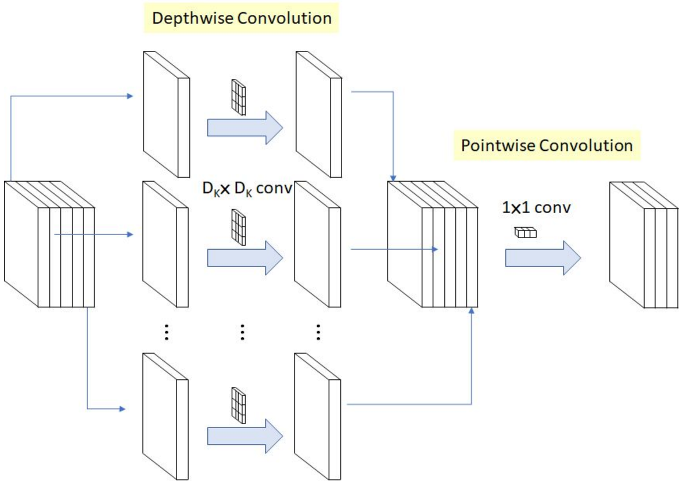
How Xception architecture looks like?

Depthwise separable + 1x1 conv

Table 1. Classification performance comparison on ImageNet (single crop, single model). VGG-16 and ResNet-152 numbers are only included as a reminder. The version of Inception V3 being benchmarked does not include the auxiliary tower.

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

MobileConv



Convolution calculations

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Xception calculations

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

$$\boxed{= \frac{1}{N} + \frac{1}{D_K^2}}$$

if we take N=512 filters and kernel size D=3 -> we get approximately 9 times less calclations

MobileNet

Table 1. MobileNet Body Architecture

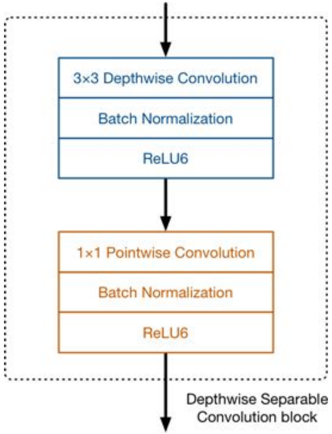
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
5× Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

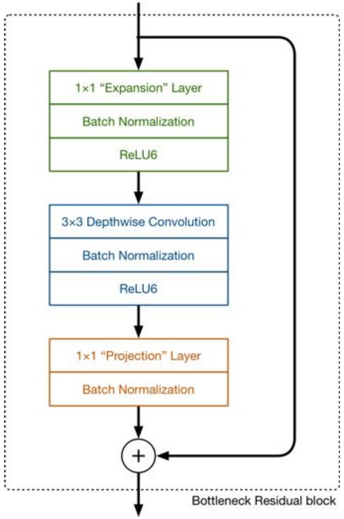
Table 8. MobileNet Comparison to Popular Models

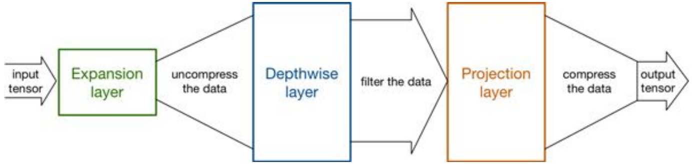
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Same quality but number of calculations and number of parameters ~1.5 times less

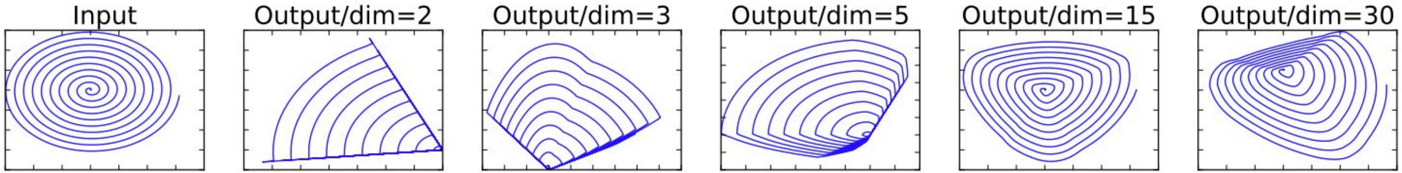
MobileNetV2





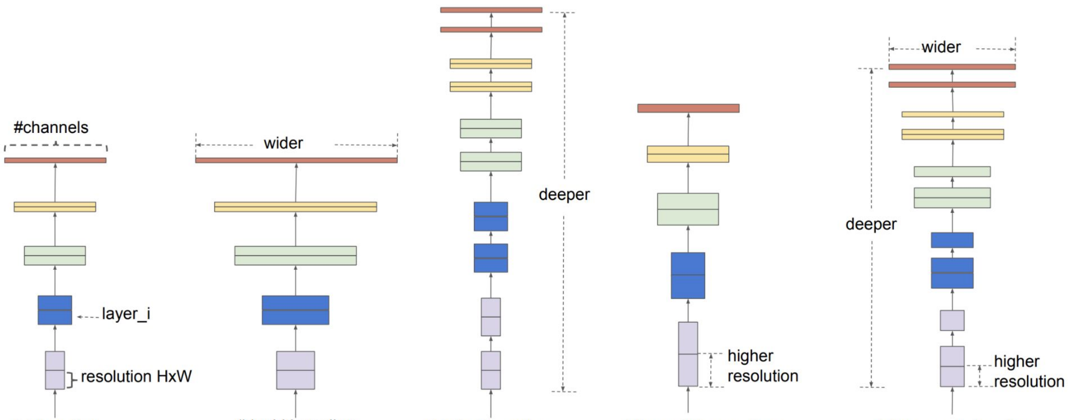


ReLU in low dimension can kill a lot of information



EfficientNet

We can improve quality of the model by increasing resolution/depth/width of the model. What is the optimal balance between them?



(a) baseline (b) width scaling (c) depth scaling (d) resolution scaling (e) compound scaling

$$\text{depth: } d = \alpha^\phi$$

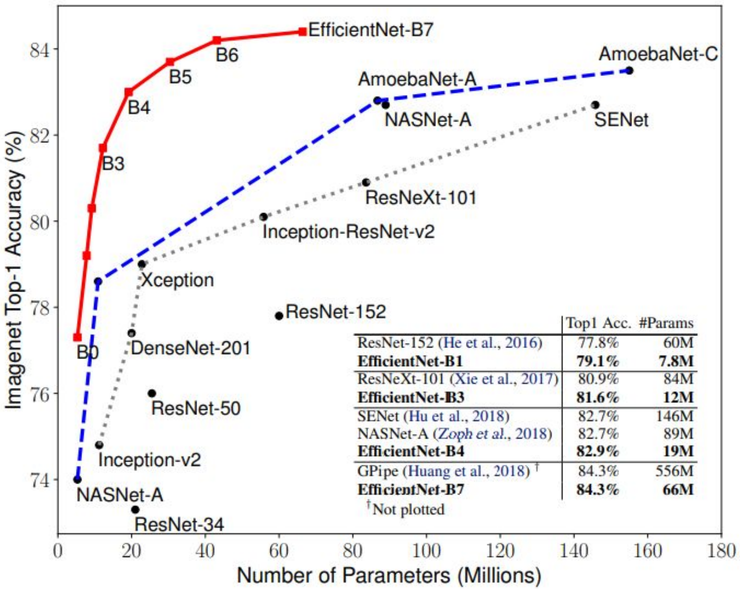
$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

EfficientNet



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

† Not plotted

EfficientNet problems

1. When large image resolution was used to train the large models, the training was slow.

2. In the early layers of the network architecture, depthwise convolutional layers (MBConv) were slow. Depthwise convolutional layers generally have fewer parameters than regular convolutional layers, but the problem is that they cannot fully make use of modern accelerators.

3. Equal scaling was applied to the height, width, and image resolution to create the various EfficientNet models from B0 to B7. This equal scaling of all layers is not optimal.

EfficientNetV2

Adding a combination of MBConv and Fused-MBConv blocks

MBConv block often cannot fully make use of modern accelerators. Fused-MBConv layers can better utilize server/mobile accelerators.

NAS search to optimize Accuracy, Parameter Efficiency, and Training Efficiency

Intelligent Model Scaling

i. maximum image size was restricted to 480x480 pixels to reduce GPU/TPU memory usage, hence increasing training speed.
ii. more layers were added to later stages, to increase network capacity without increasing much runtime overhead.

Progressive Learning
The idea is very simple. In the earlier steps, the network was trained on small images and weak regularization. This allows the network to learn the features fast. Then the image sizes are gradually increased, and so are the regularizations.

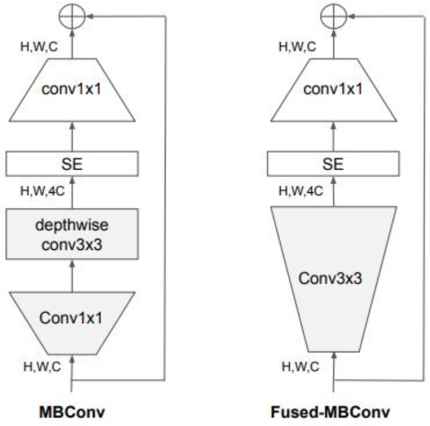


Figure 2. Structure of MBConv and Fused-MBConv.

ViT: Visual Transformer

https://nn.labml.ai/transformers/vit/index.html

ViT: Patching

How to adapt transformer for computer vision? First thing to deal with is to attention. If we consider one pixel as a token we have our attention matrix will have size (500x500) ^ 2 which is huge. So, the authors proposed to slice image on patches and work with them.

2) Unfold

```python
from einops import rearrange

p = patch_size # P in maths

x_p = rearrange(img, 'b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = p, p2 = p)
```

1) Slice

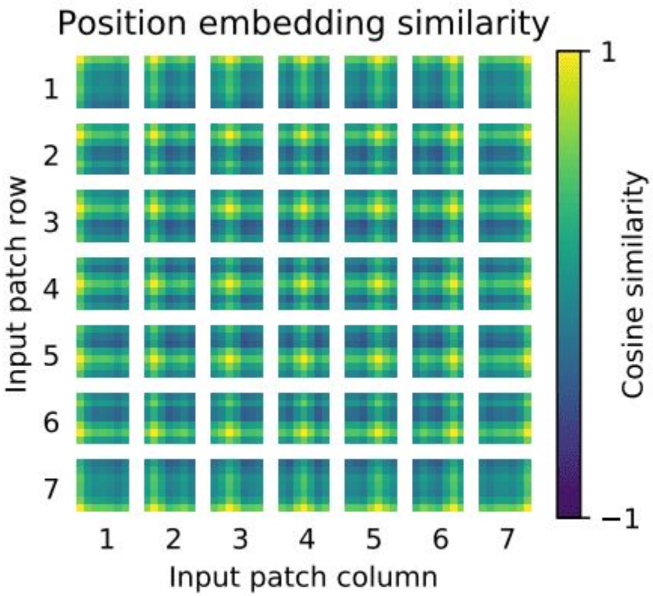
Code to implement this operation

ViT: Positional embeddings

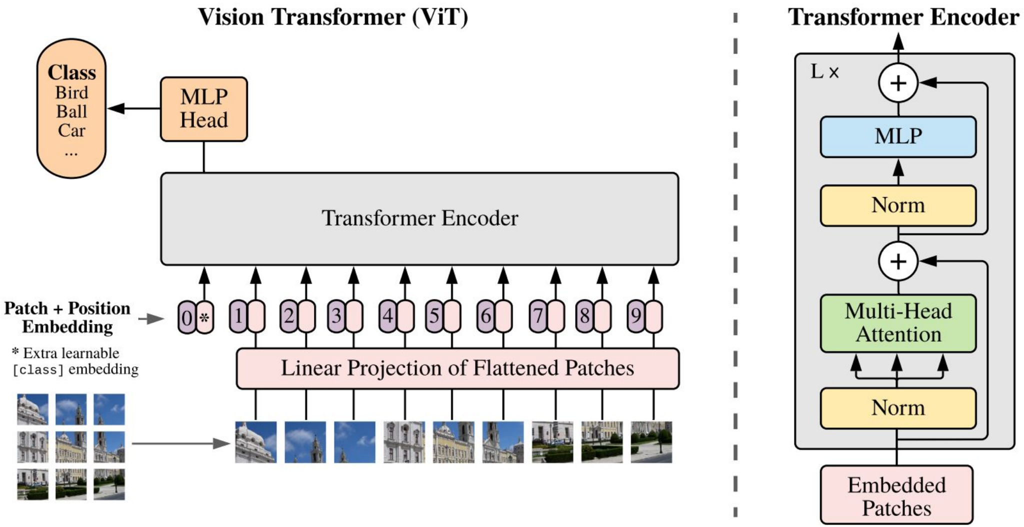
As in transformers in NLP we have to add positional information to embeddings. But the images have 2D positional information (x, y).

Authors tried both 2D positional and sum of two 1D positional embeddings. The latter works with the same quality. And indeed the learned positional matrix looks at neighbourhood pixels.

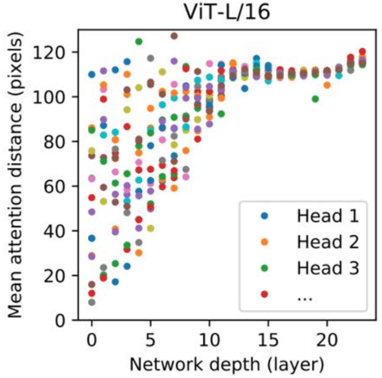
To fine-tune in higher resolutions, 2D interpolation of the pre-trained position embeddings is performed.

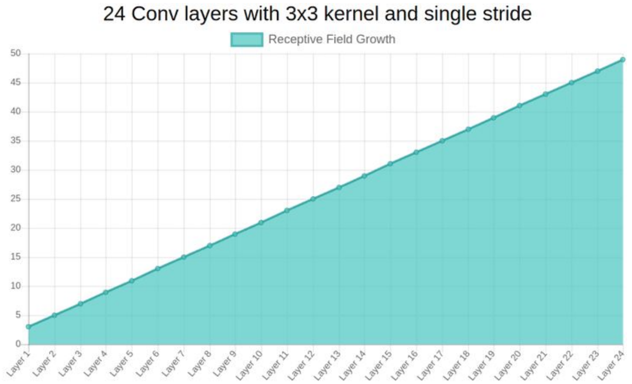


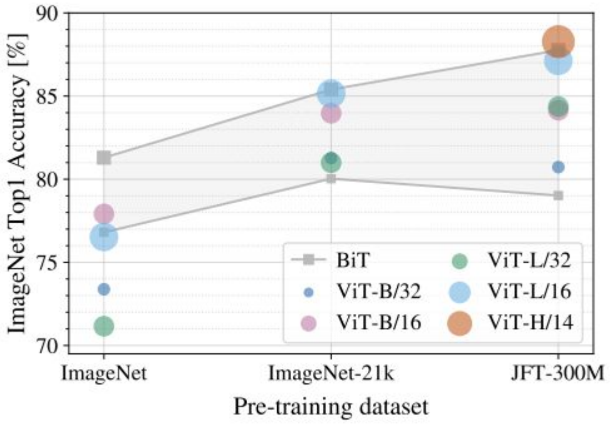
ViT: architecture



ViT: benefits







1) ViT has a global perception field in contrast of CNN. And from the figure we can see what model start to use this opportunity from the beginning

2) ViT continues to scale with data, CNN doesn't

Model Zoo: Where to find a model for your task

Models
huggingface.co/models

https://huggingface.co/models

timm
Documentation

https://huggingface.co/docs/timm/index

PyTorch

https://pytorch.org/vision/stable/models.html#classification

Recap

- AlexNet
- VGG
- Inception
- ResNet
- Xception
- MobileNet
- EfficientNet
- ViT