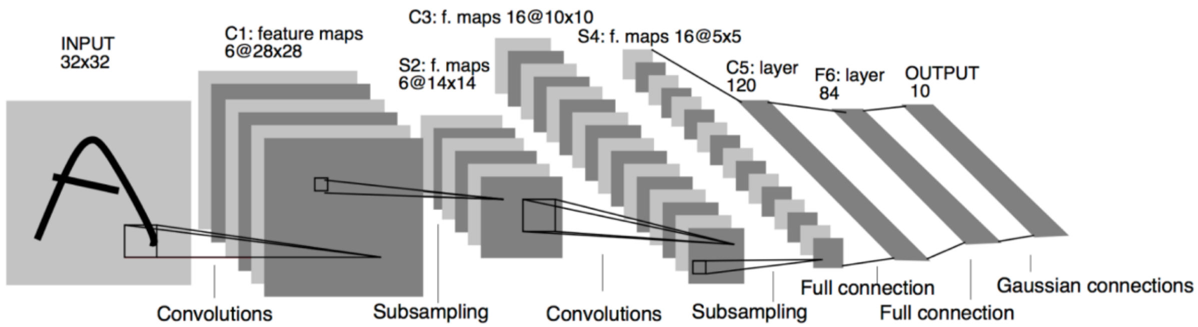
Deep Learning – some popular architectures and history

Bernhard Kainz

LeNet-5

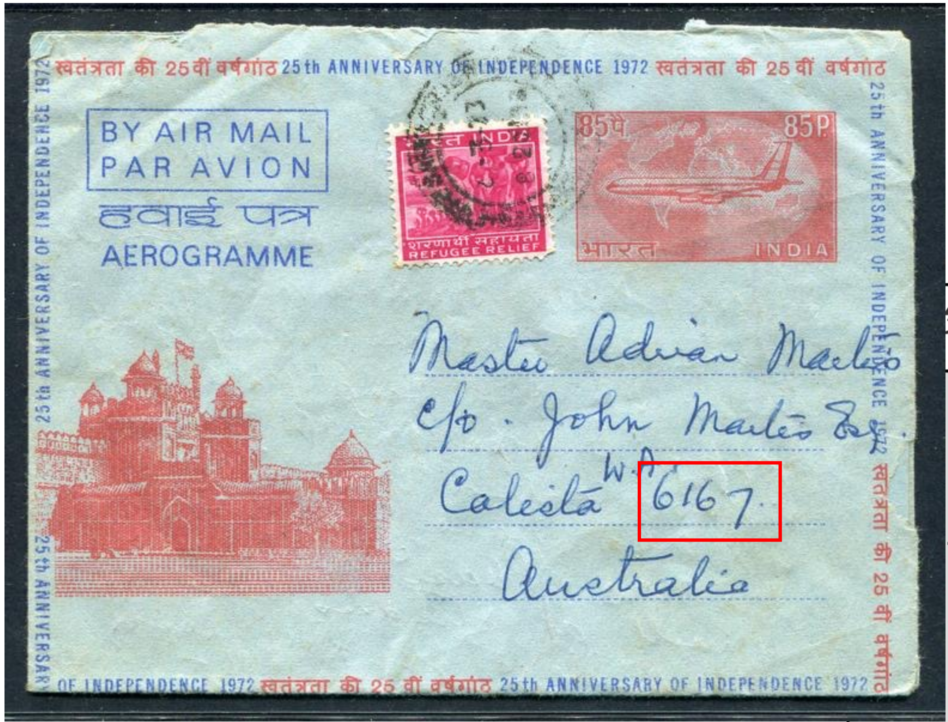
Gradient-Based Learning Applied to Document Recognition

YANN LECUN, MEMBER, IEEE, LÉON BOTTOU, YOSHUA BENGIO, AND PATRICK HAFFNER



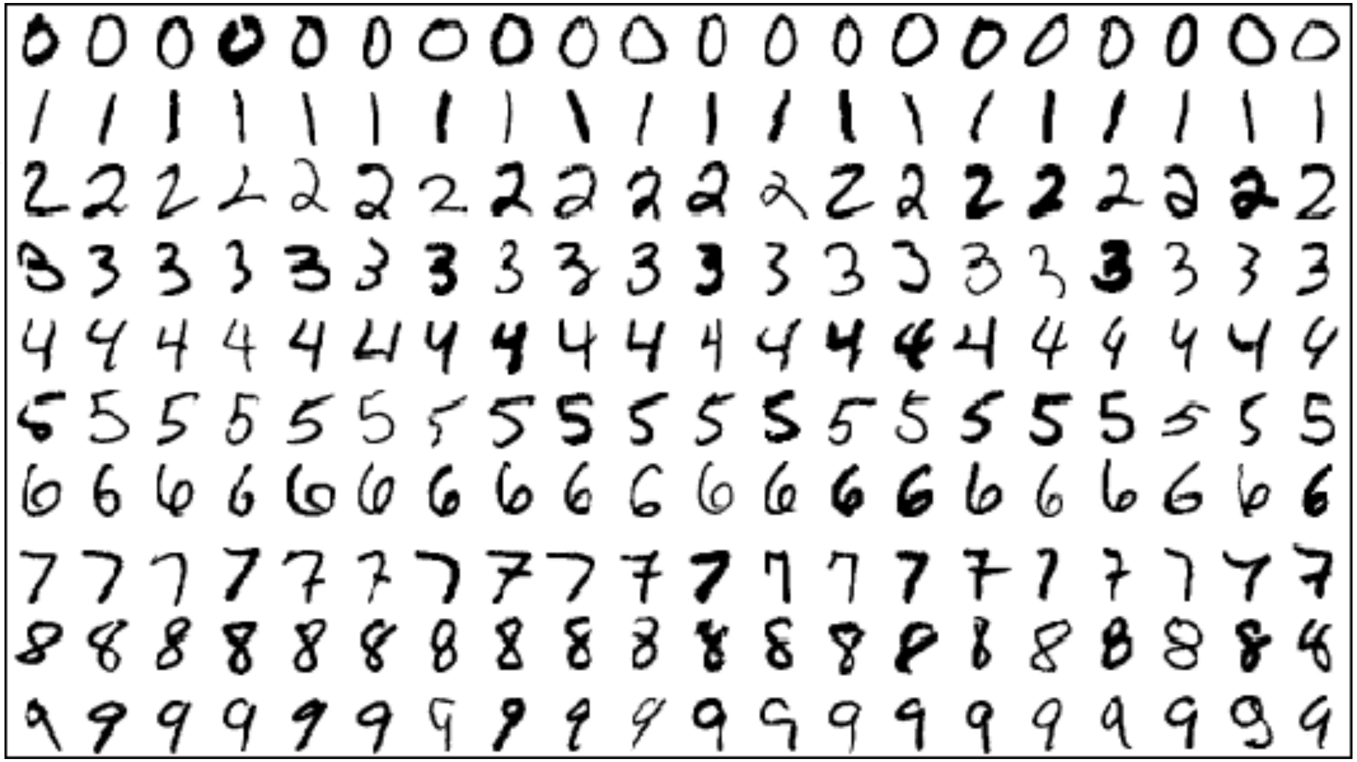
LeCun et al. 1998

Handwritten digit recognition



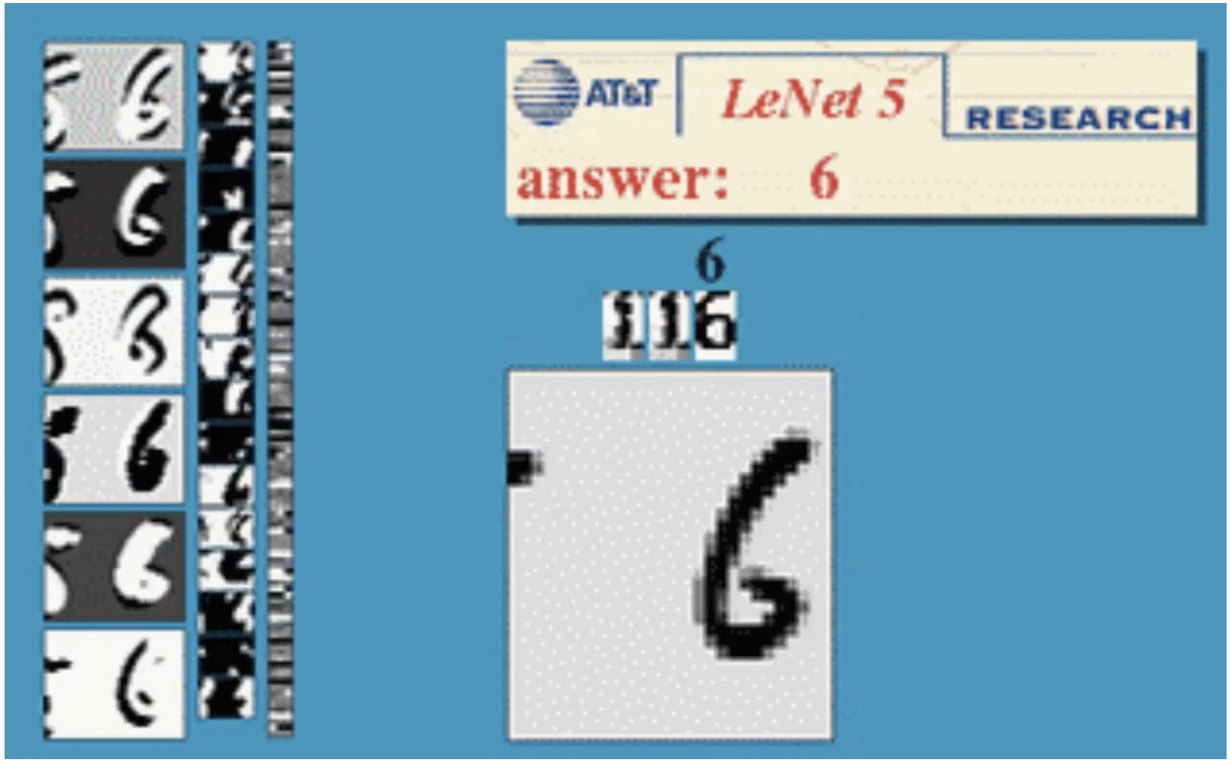
MNIST

- Cantered and scaled
- 50.000 training samples
- 10.000 test samples
- 28 x 28 images
- 10 classes



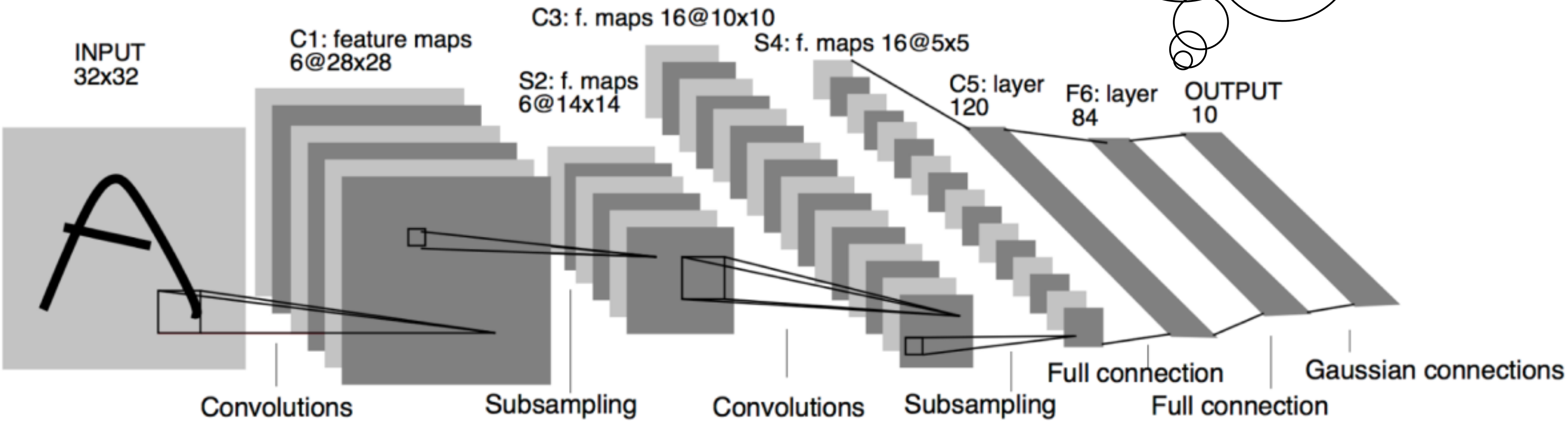
Demo from 1995

https://www.youtube.com/watch?v=yxuRnBEczUU

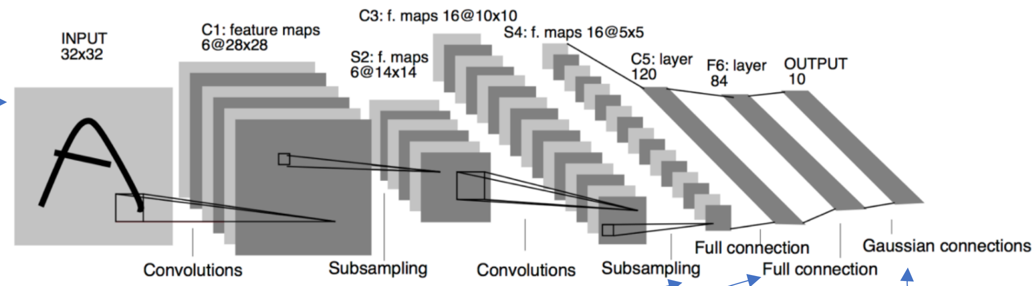


LeNet-5

This is expensive if you have many outputs, here only 10



```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features


net = Net()
print(net)
```



https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

AlexNet

AlexNet

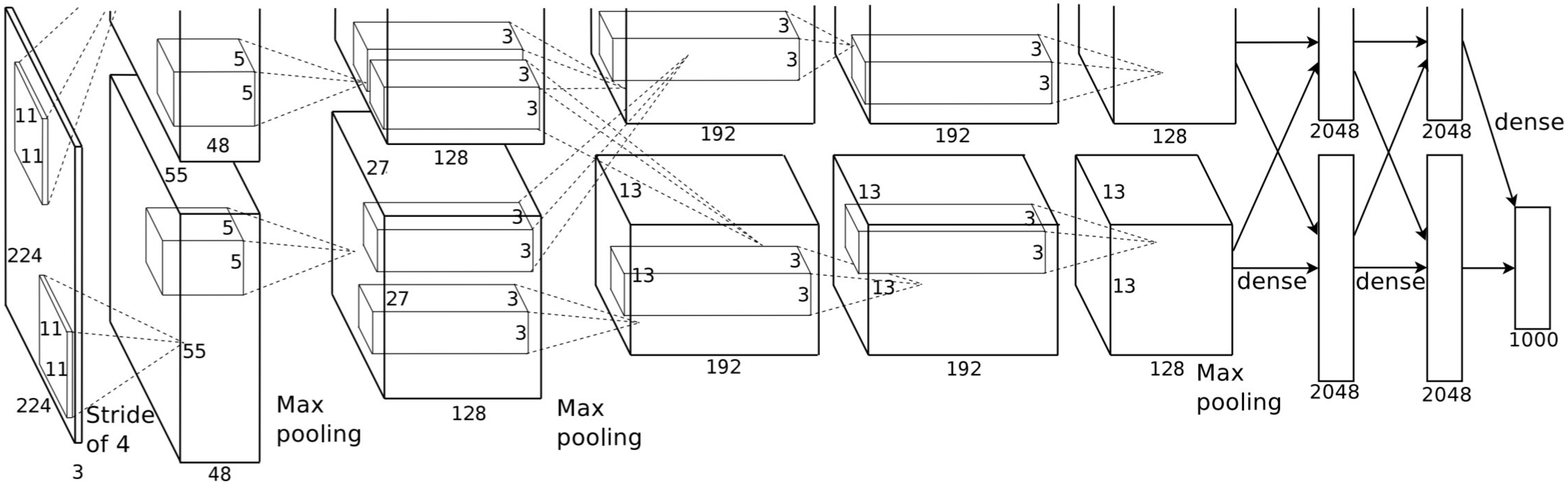


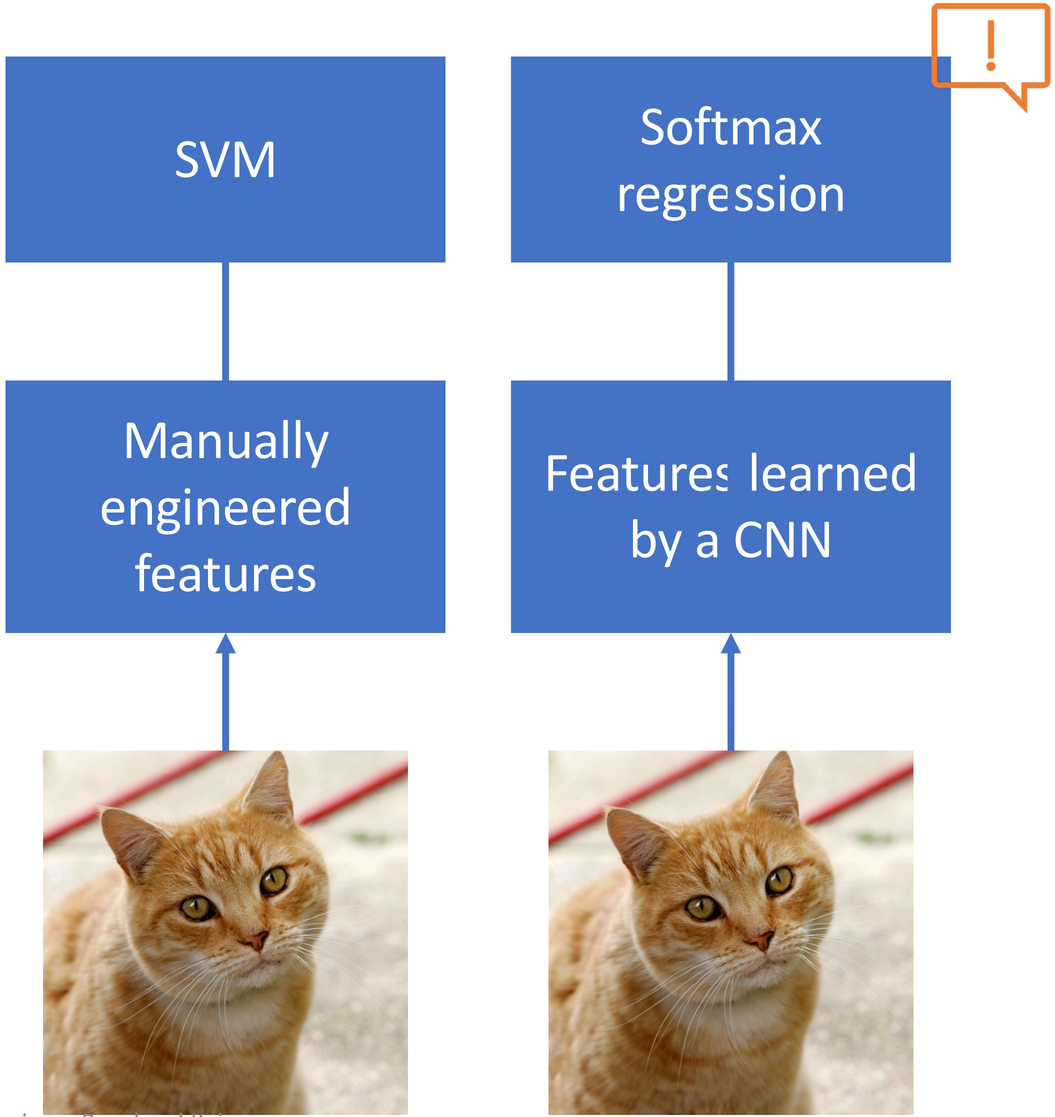
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

ImageNet (2010)

Images	Color images with nature objects	Gray image for hand-written digits
Size	469 x 387	28 x 28
# examples	1.2 M	60 K
# classes	1,000	10

AlexNet

- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications:
  - Add a dropout layer after two hidden dense layers
    (better robustness / regularization)
  - Change activation function from sigmoid to ReLu
    (no more vanishing gradient)
  - MaxPooling
  - Heavy data augmentation
  - Model ensembling
- Paradigm shift for computer vision

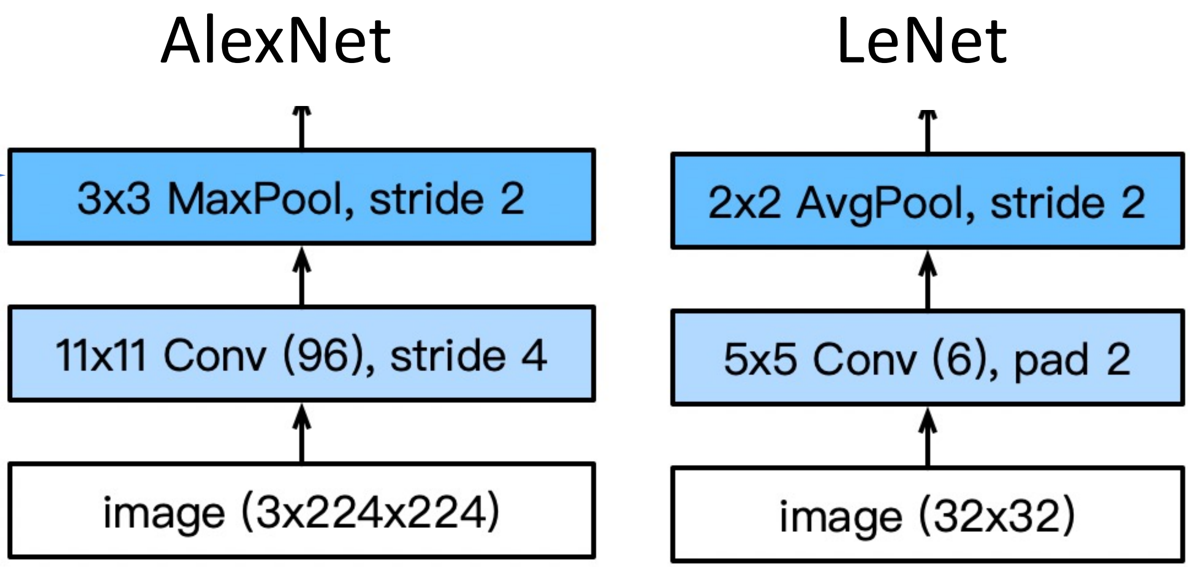


Slide adopted from Alex Smola

AlexNet Architecture

Larger pool size, change to max pooling

Larger kernel size, stride becaus
of the increased image size, an
more output channels.

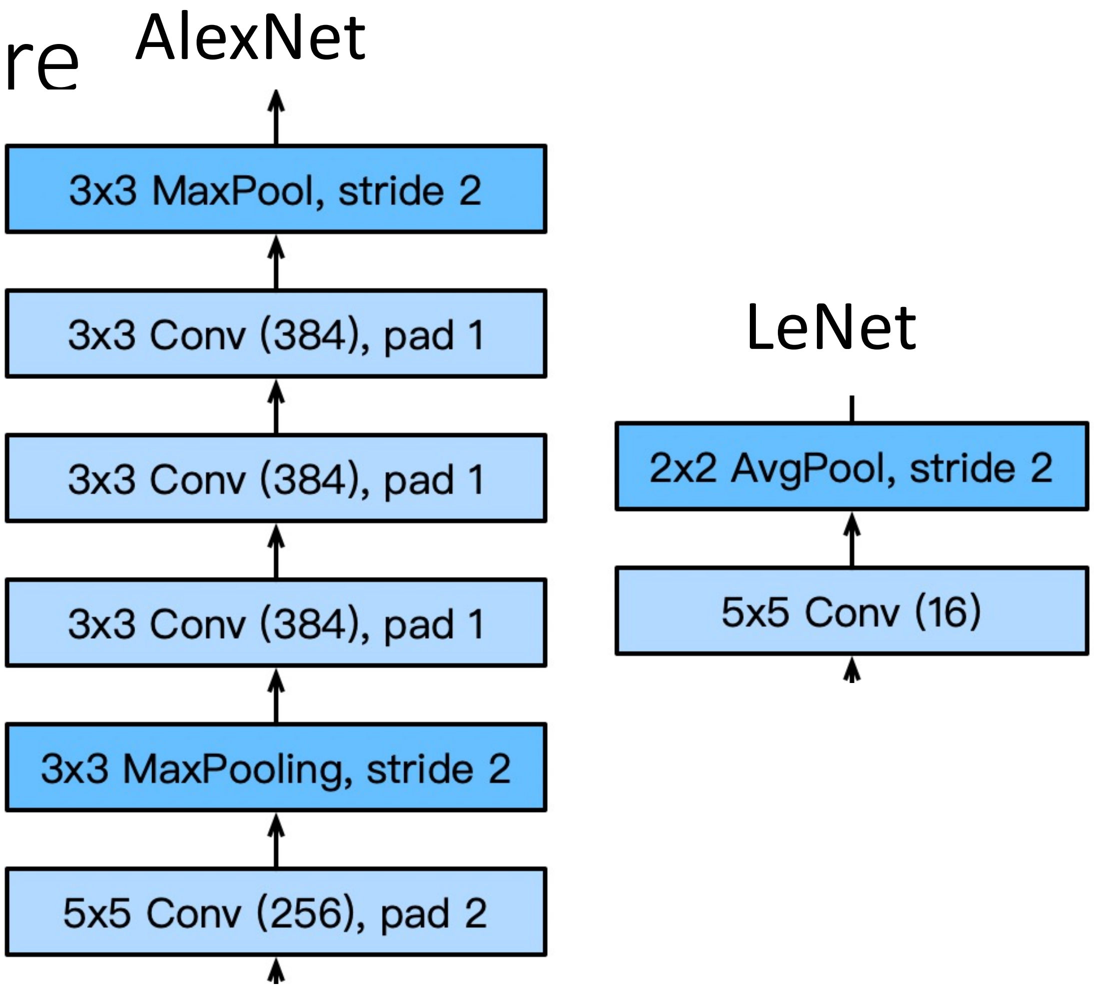


Slide adopted from Alex Smola

AlexNet Architecture

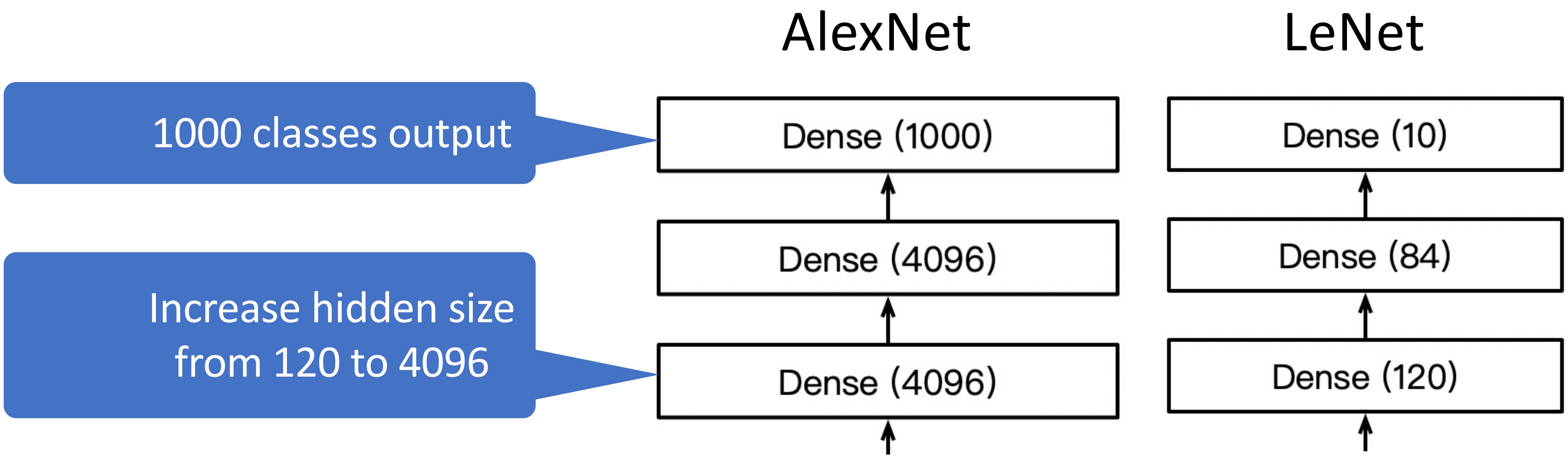
3 additional convolutional layers

More output channels.



Slide adopted from Alex Smola

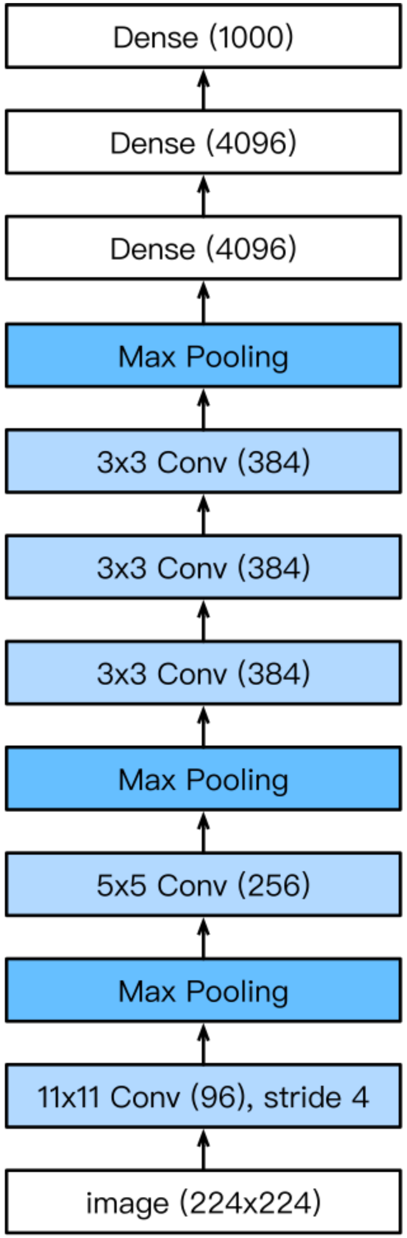
AlexNet Architecture



Slide adopted from Alex Smola

Complexity

	#parameters		FLOP	
	AlexNet	**LeNet**	**AlexNet**	**LeNet**
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x

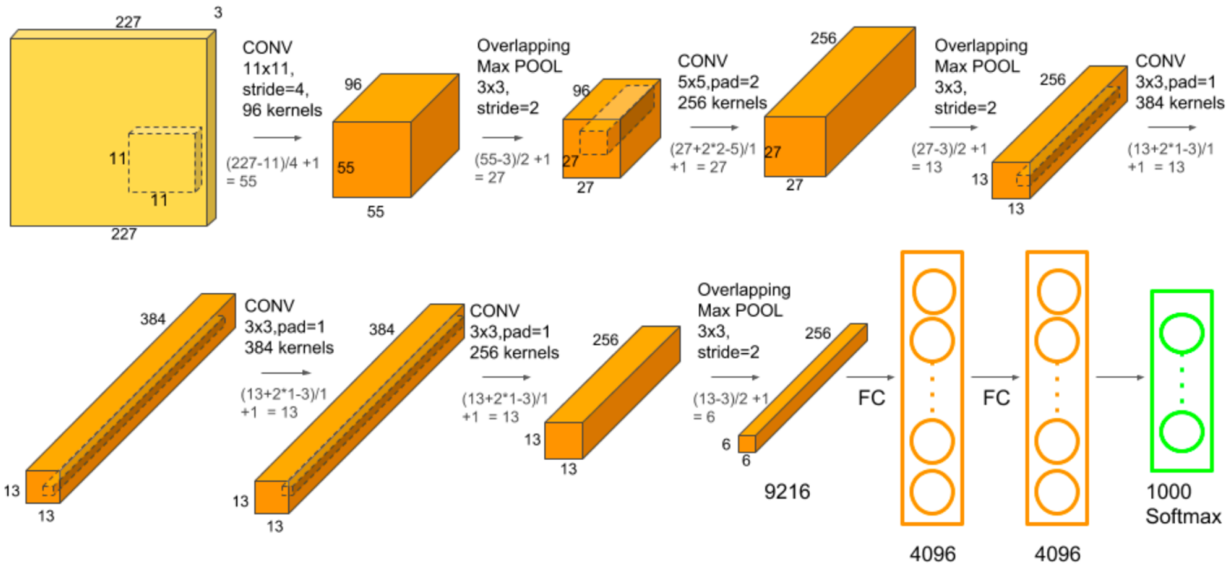


Slide adopted from Alex Smola

demo

Silicon Valley: Season 4 Episode 4: Not Hotdog (HBO)
https://www.youtube.com/watch?v=pqTntG1RXSY

```python
class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

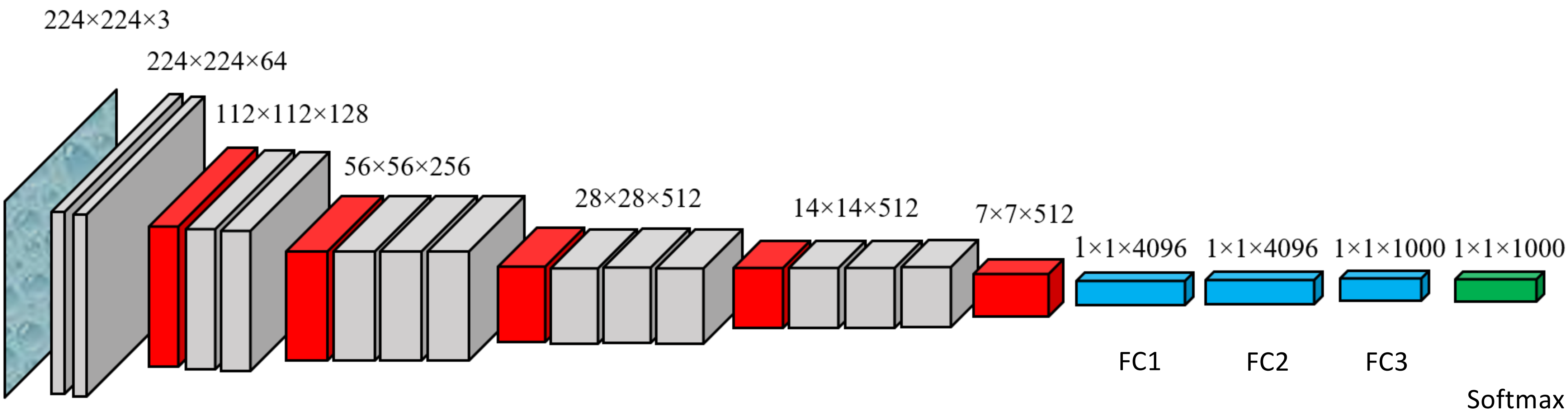


https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py

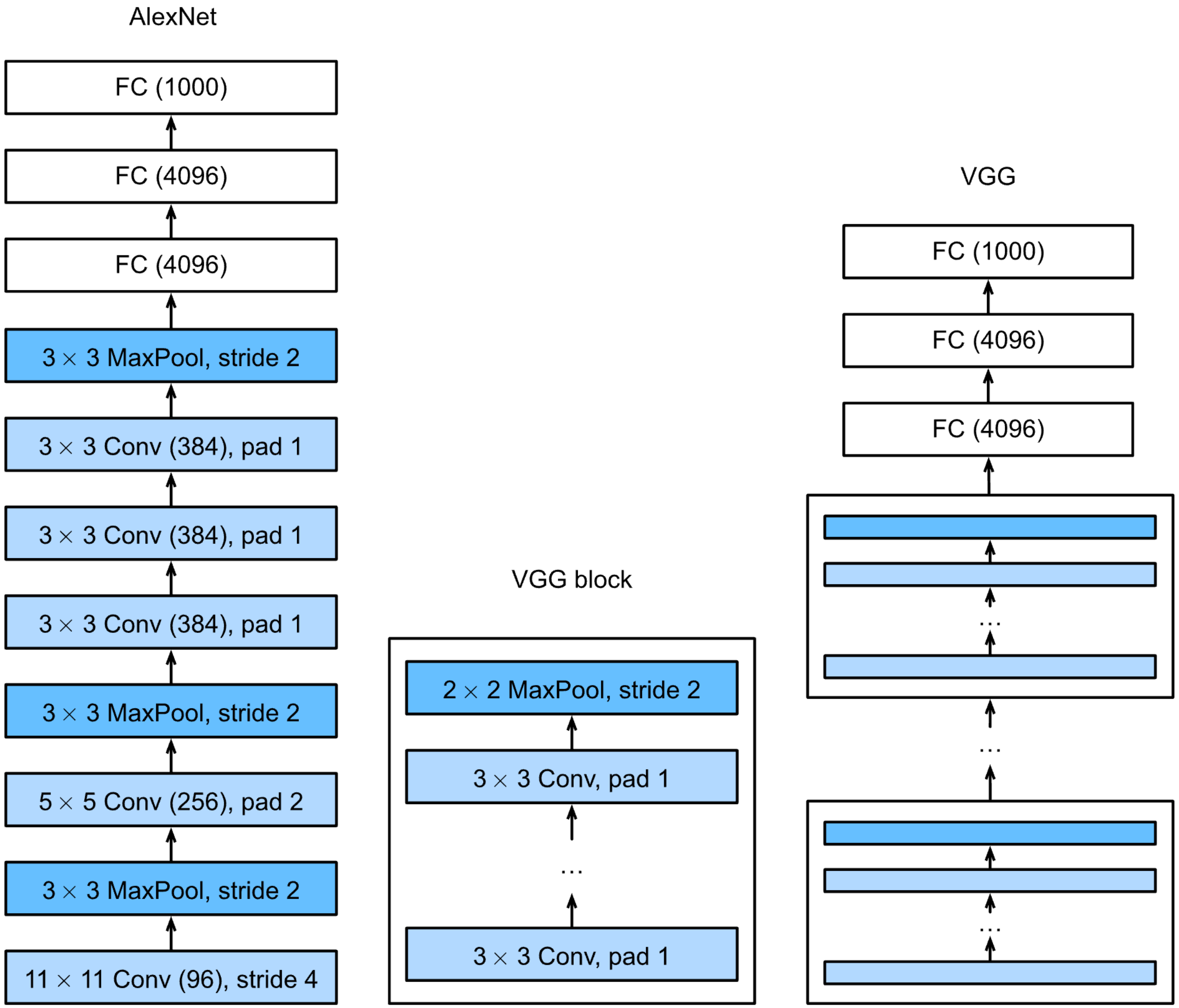
VGG

Lecture inspired by Alex Smola with add-ons

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
1×1×4096
1×1×1000
1×1×1000
FC1
FC2
FC3
Softmax

- AlexNet = bigger than LeNet
- Bigger = better?
- Options
  - **More** dense layers (too expensive)
  - **More** convolutions
  - Group into **blocks**

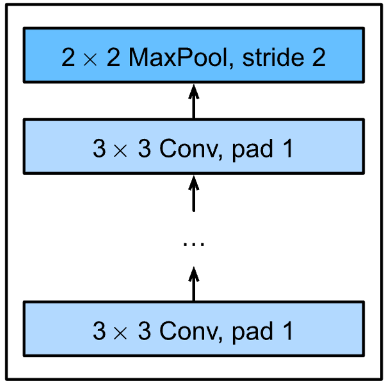


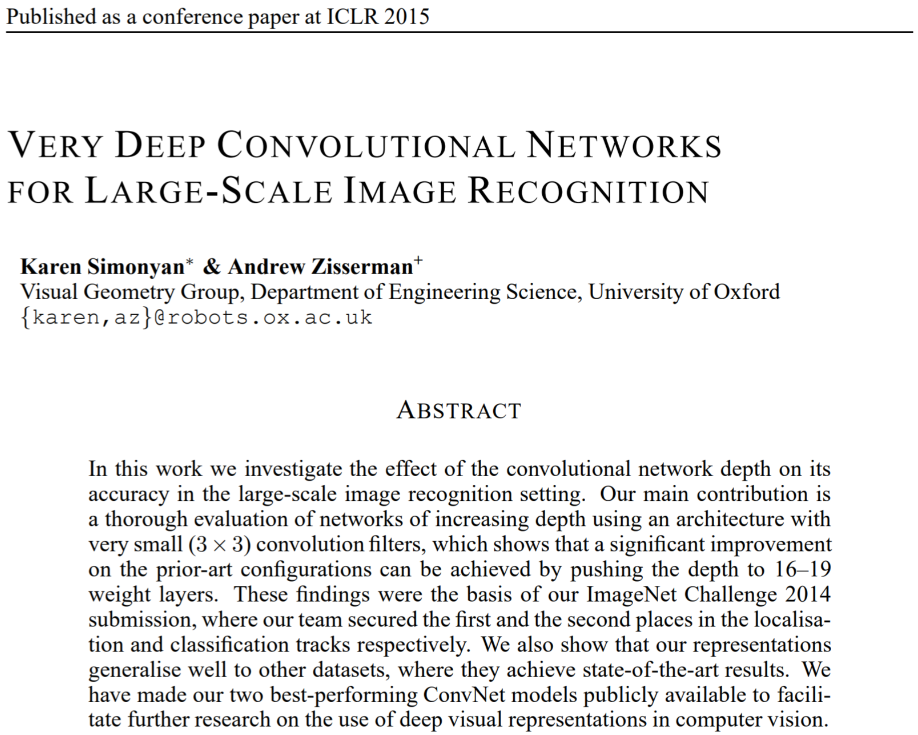
http://d2l.ai/chapter_convolutional-modern/vgg.html

VGG blocks

- Deeper vs. wider?
  - 13x13?
  - 5x5?
  - 3x3?
  - Deep and narrow = better
- VGG block
  - 3x3 convolutions (pad 1)
    (n layers, m channels)
  - 2x2 max-pooling
    (stride 2)

VGG block




Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman**+
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

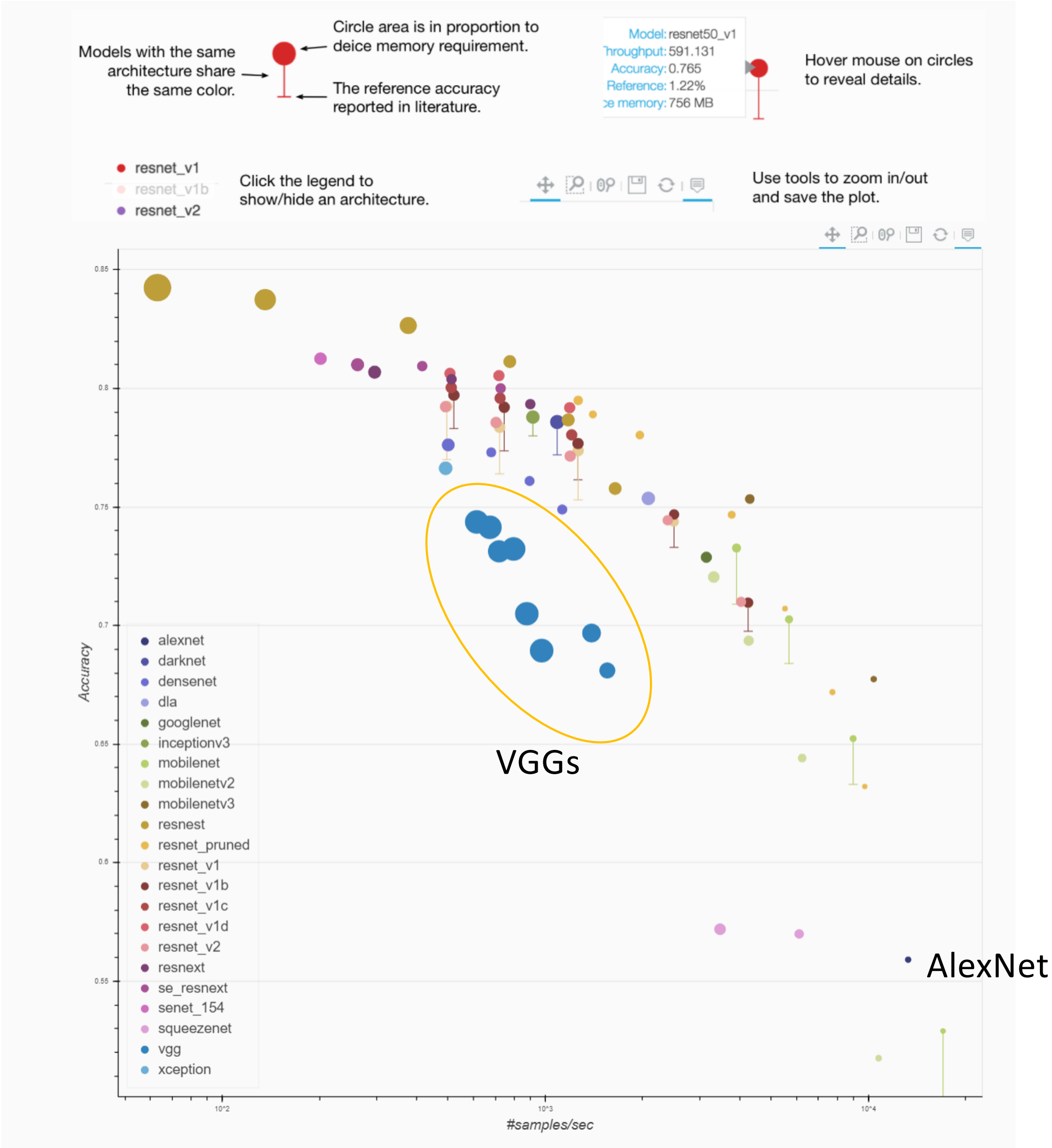
ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

progress

Wouldn’t have been possible without compute power progress (GPUs)

- LeNet (1995)
  - 2 convolution + pooling layers
  - 2 hidden dense layers
- AlexNet
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- VGG
  - Bigger and deeper AlexNet (repeated VGG blocks)



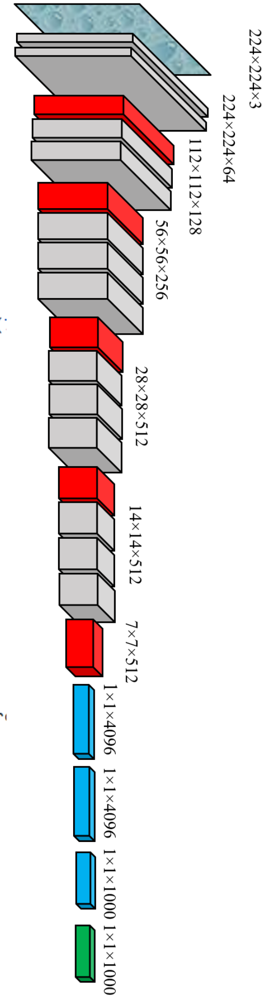
https://cv.gluon.ai/model_zoo/classification.html

```python
def vgg16(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> VGG:
    r"""VGG 16-layer model (configuration "D")
    `"Very Deep Convolutional Networks For Large-Scale Image Recognition" <https://arxiv.org/pdf/1409.1556.pdf>`_

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _vgg('vgg16', 'D', False, pretrained, progress, **kwargs)
```

```python
def _vgg(arch: str, cfg: str, batch_norm: bool, pretrained: bool, progress: bool, **kwargs: Any) -> VGG:
    if pretrained:
        kwargs['init_weights'] = False
    model = VGG(make_layers(cfgs[cfg], batch_norm=batch_norm), **kwargs)
    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[arch],
                                              progress=progress)
        model.load_state_dict(state_dict)
    return model
```

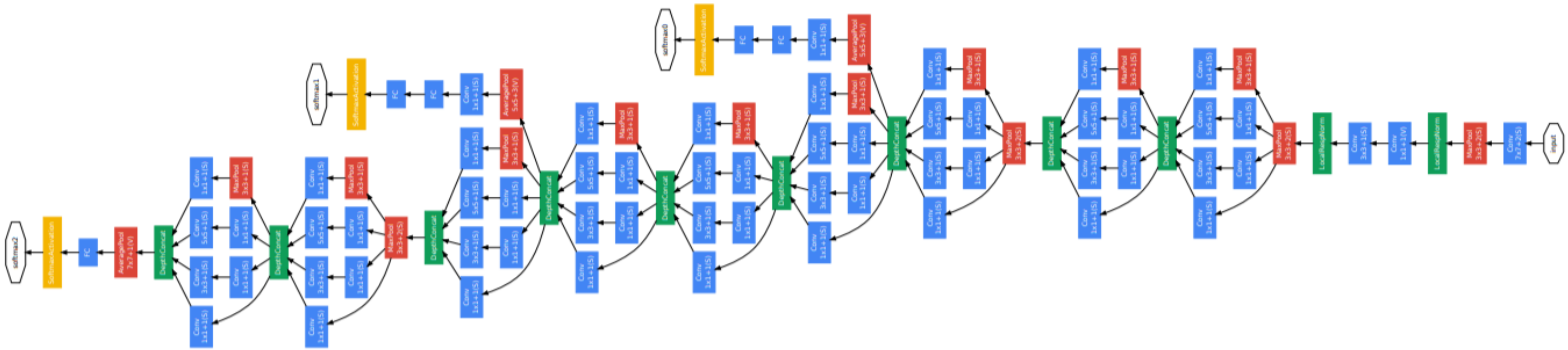
```python
def make_layers(cfg: List[Union[str, int]], batch_norm: bool = False) -> nn.Sequential:
    layers: List[nn.Module] = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            v = cast(int, v)
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

```python
class VGG(nn.Module):

    def __init__(
        self,
        features: nn.Module,
        num_classes: int = 1000,
        init_weights: bool = True
    ) -> None:
        super(VGG, self).__init__()
        self.features = features
        self.avgpool = nn.AdaptiveAvgPool2d((
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x: torch.Tensor) -> tor
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```



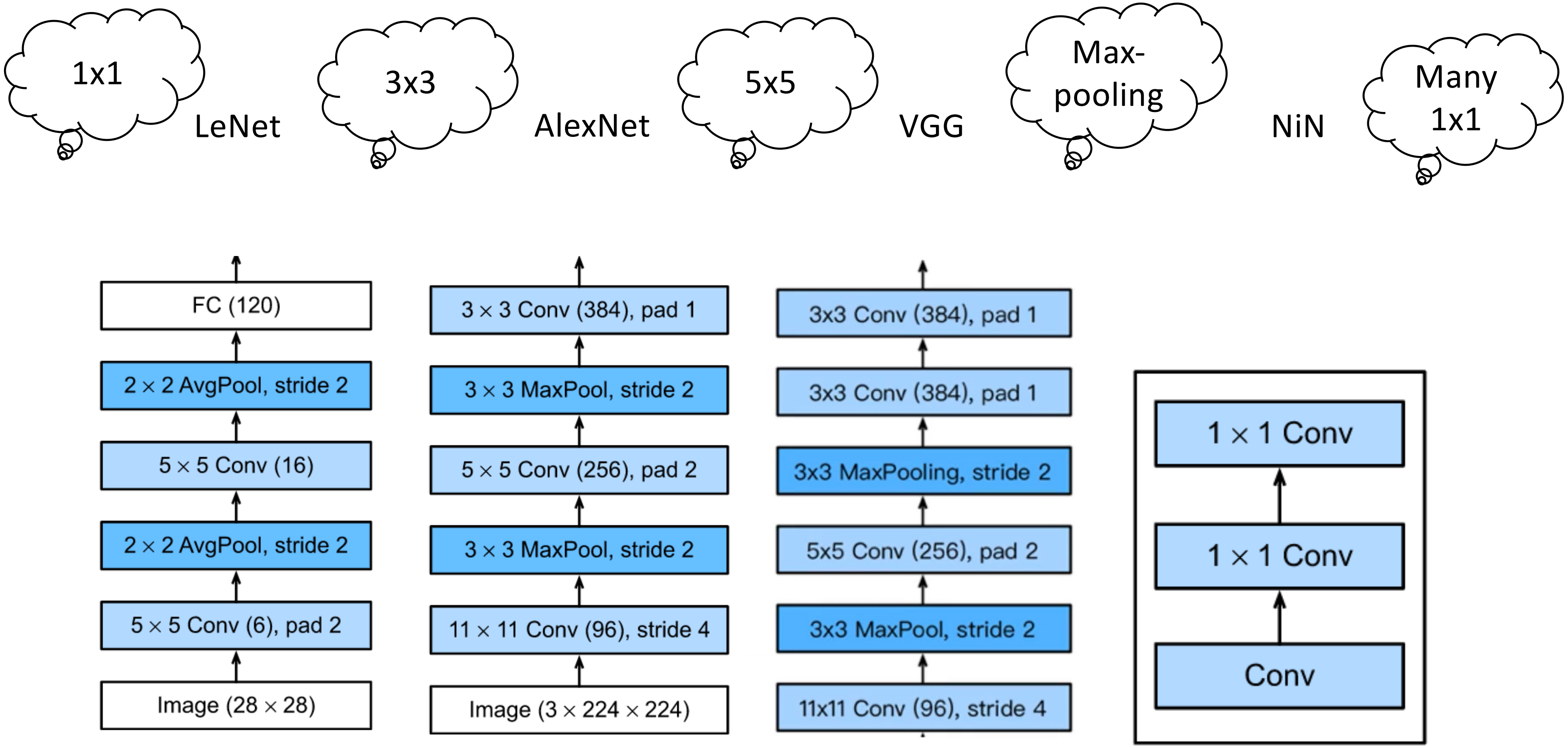
https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py

Inception (GoogLeNet)

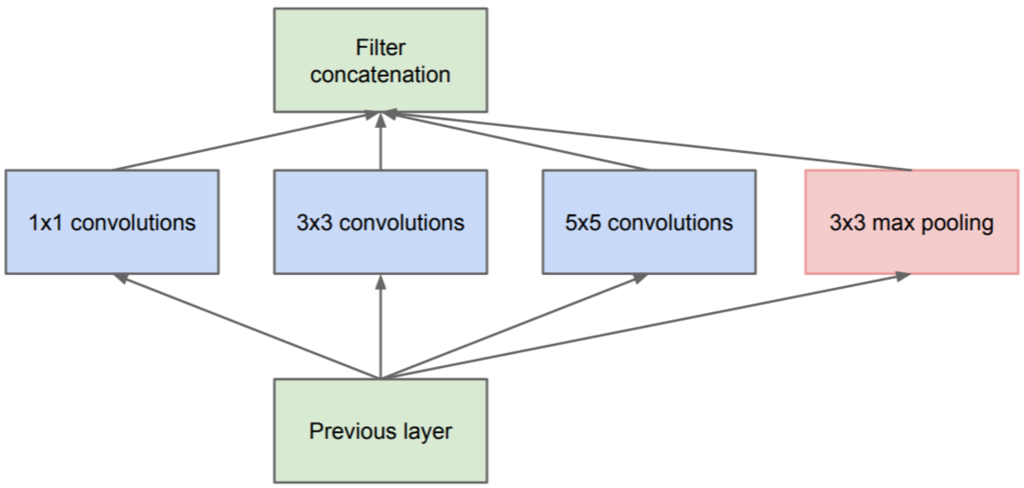


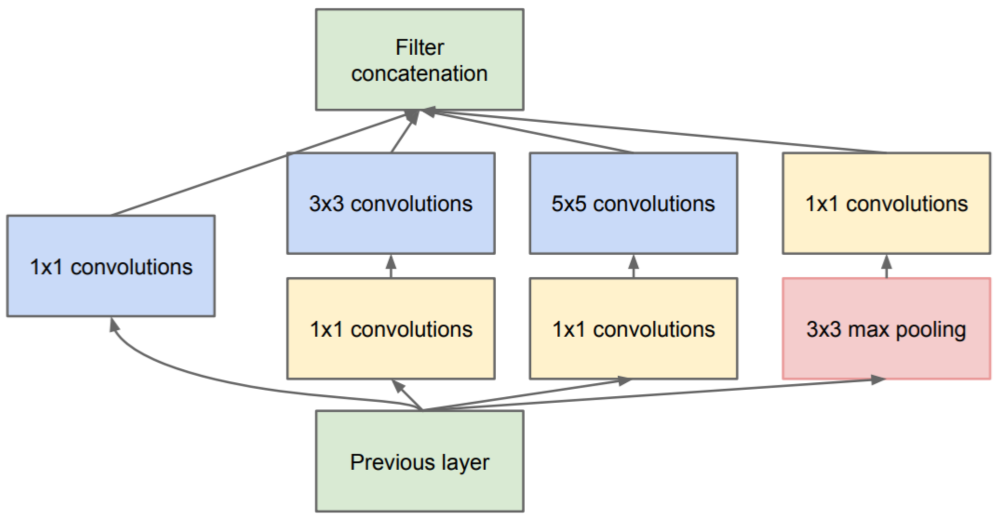
https://arxiv.org/abs/1409.4842

Which convolution is the best!?



Inception block





https://arxiv.org/abs/1409.4842

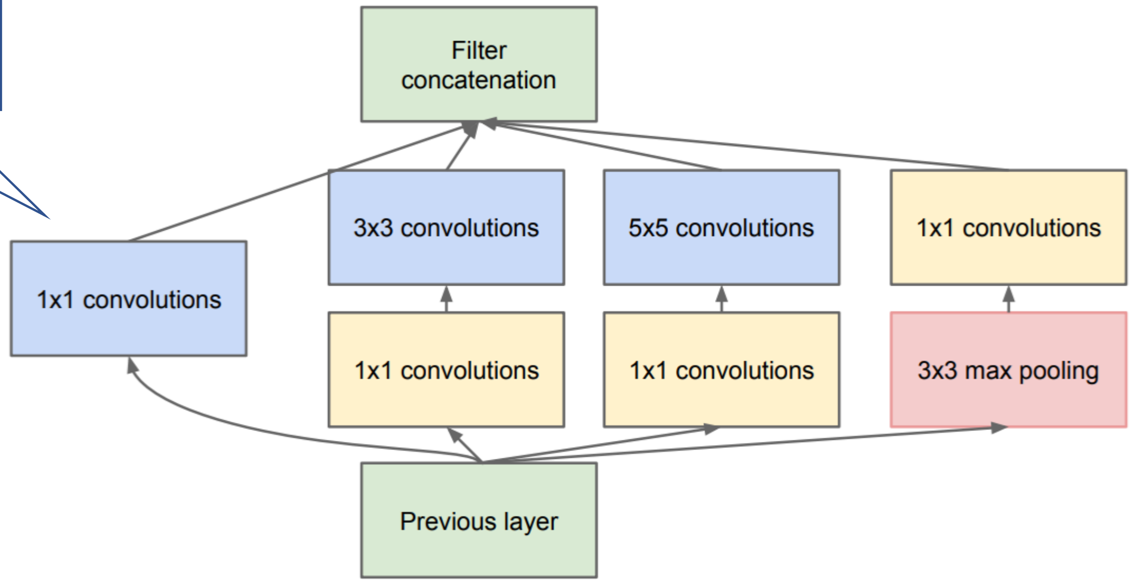
Inception block

Same width and height as input

Four paths extract information from different aspects, then concatenate along the output channel

Extract with different size convolutions

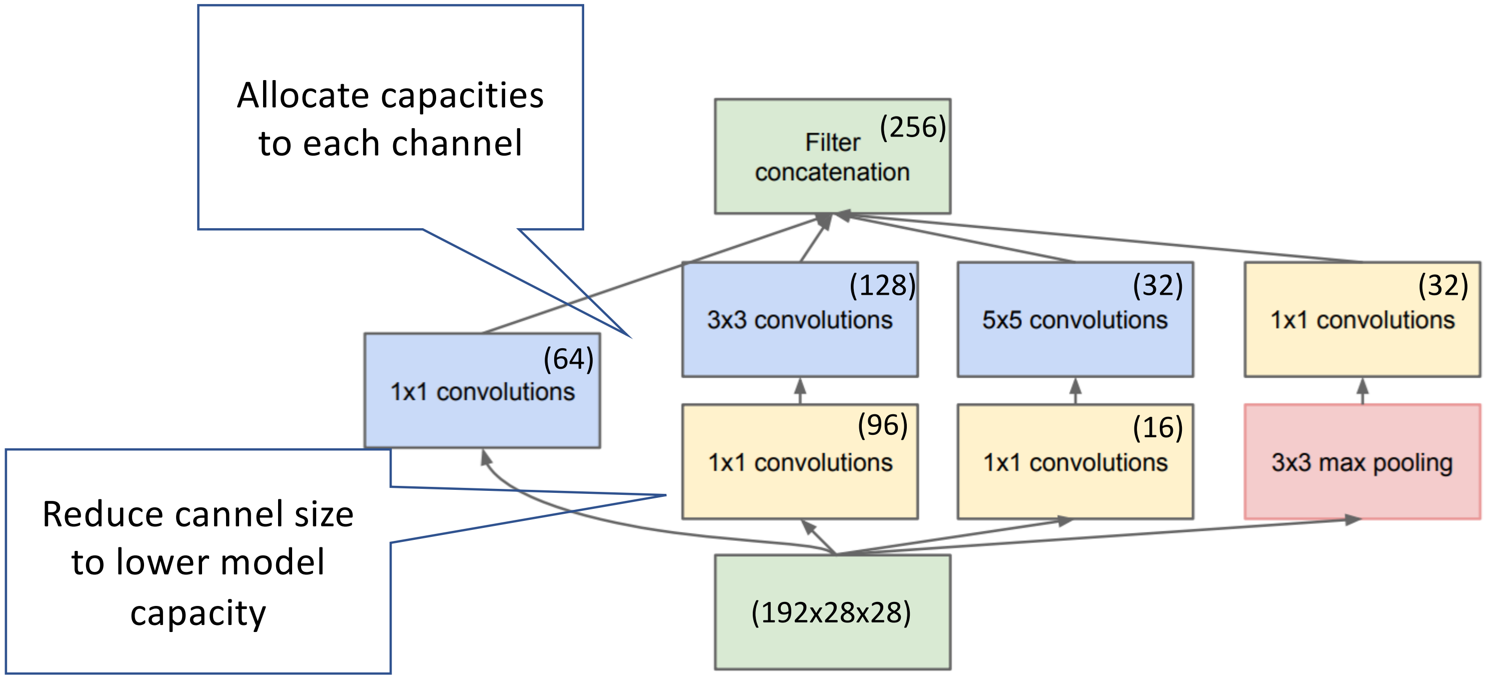
Extract spatial information with pooling



https://arxiv.org/abs/1409.4842

Inception block

The first inception block has channel sizes specified



https://arxiv.org/abs/1409.4842

Inception blocks

- Inception blocks have fewer parameters and less computation complexity than single 3x3 or 5x5 convolution layers
- They are a mix of different functions, which makes them a powerful function class
- Computing and memory wise they are efficient (good generalisation)

	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M

As: replace all conv block with 3x3 or 5x5 in Inception

Less operations?

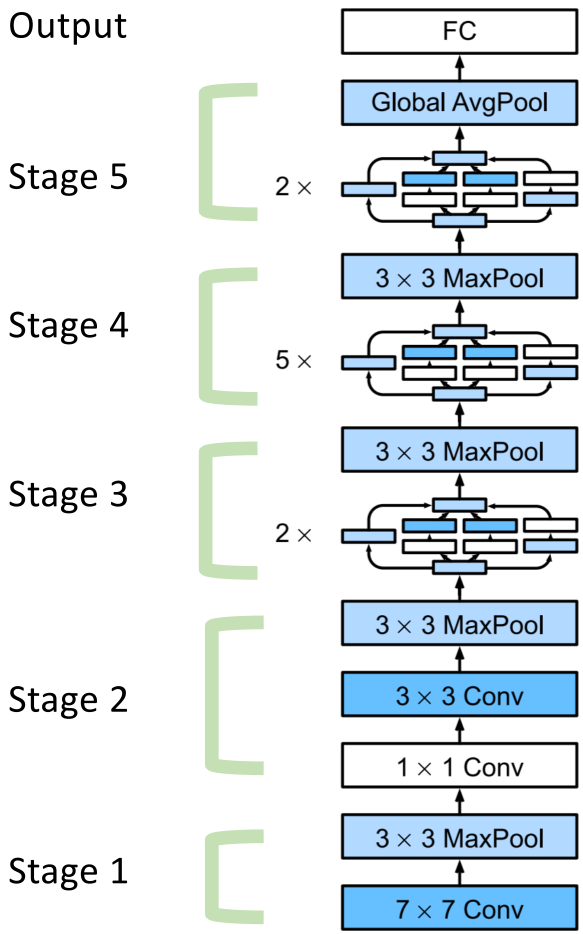
- $k^2 \times \underbrace{\boxed{c_{in}}}_{\text{fixed}} \times c_{out} \times m_h \times \underbrace{\boxed{m_w}}_{\text{fixed}}$

- $c_{in} \times m_h \times m_w \times [\, \sum_{paths\ j} k_j^2 \times c_{out,j}\,]$

allocating compute
to different channels
= better computing

Inception

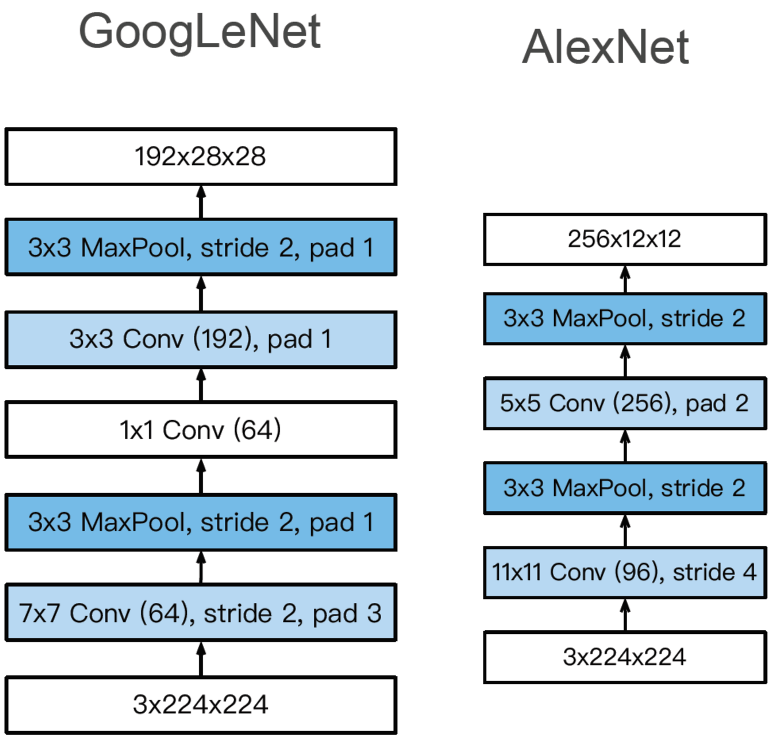
- 5 stages with 9 inception blocks



https://d2l.ai/

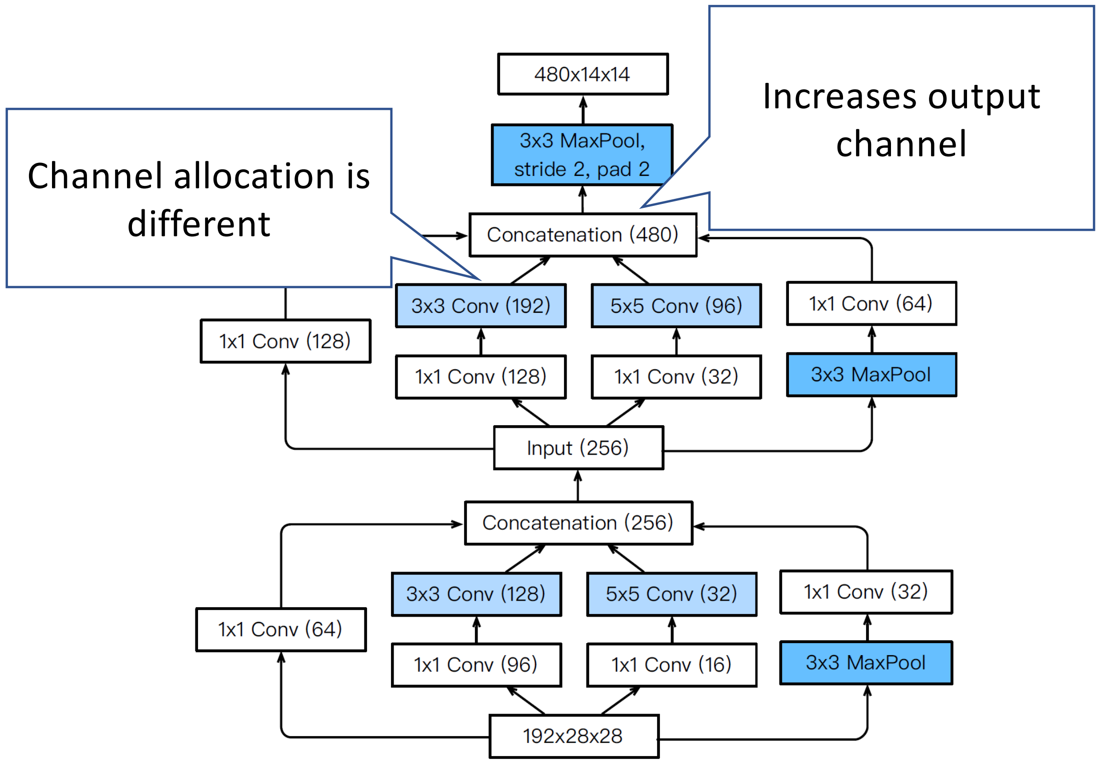
Stage 1 and 2

- Smaller kernel size and output channels because of more layers



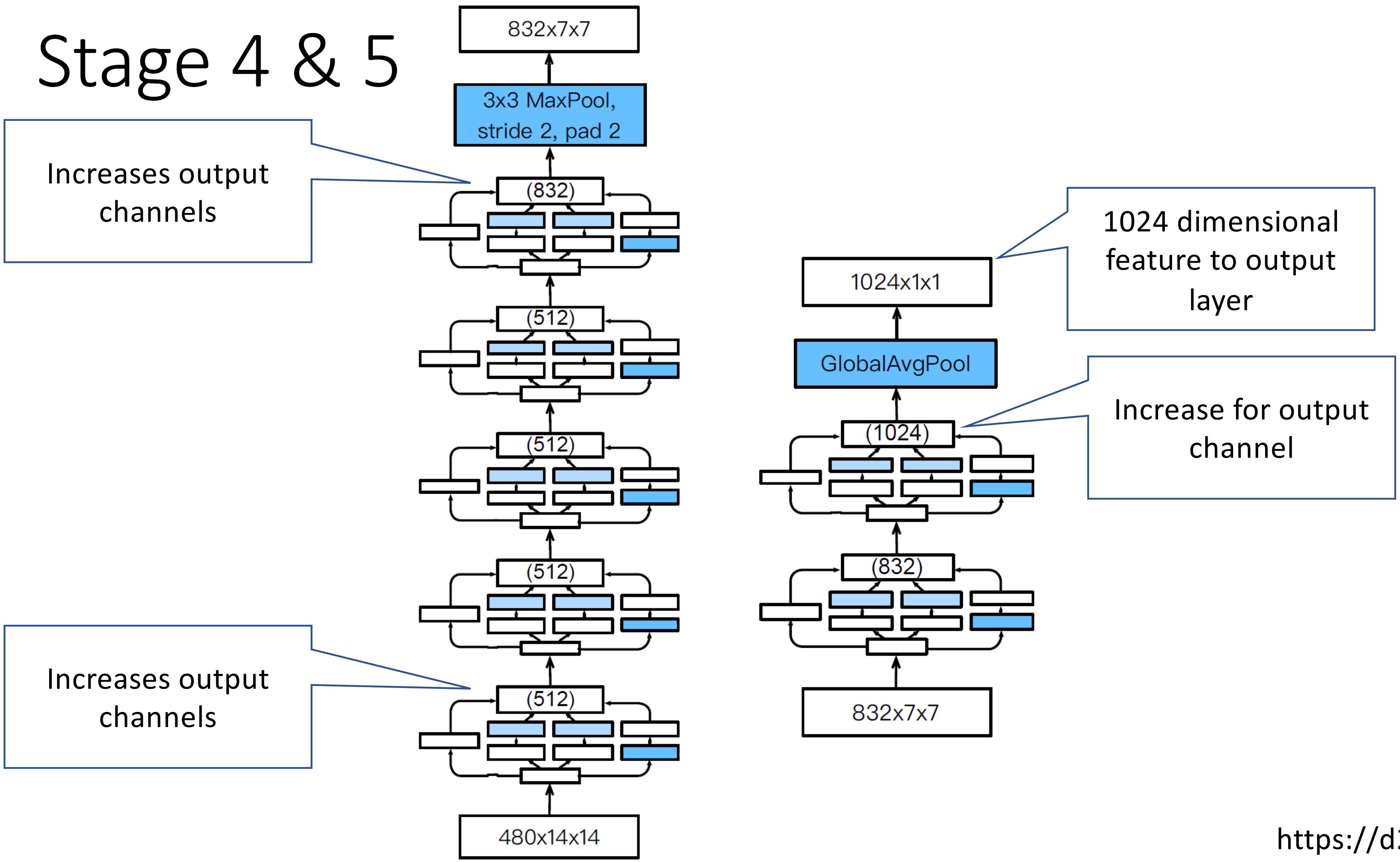
https://d2l.ai/

Stage 3



https://d2l.ai/

Stage 4 & 5

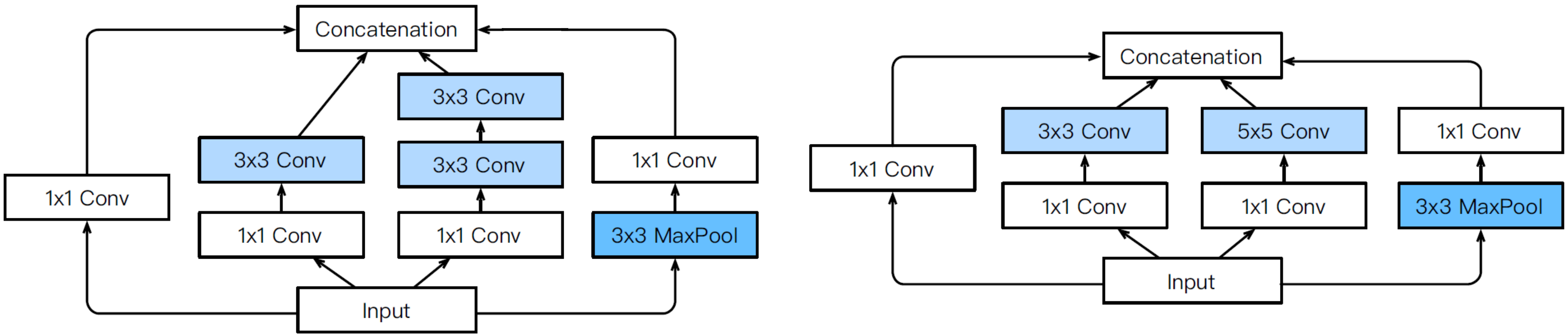


https://d2l.ai/

Flavours of Inception Networks

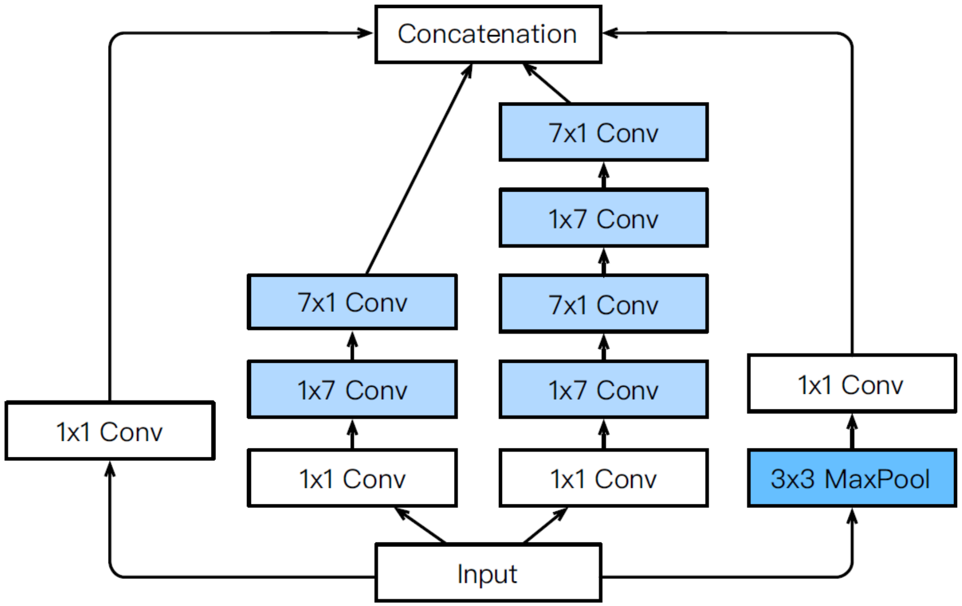
- Inception-BN (v2) – added batch normalisation
- Inception-V3 – Modified the inception block
  - Replace 5x5 by multiple 3x3 convolutions
  - Replace 5x5 by 1x7 and 7x1 convolutions
  - Replace 3x3 by 1x3 and 3x1 convolutions
  - Generally deeper stack
- Inception-V4 – adds residual connections

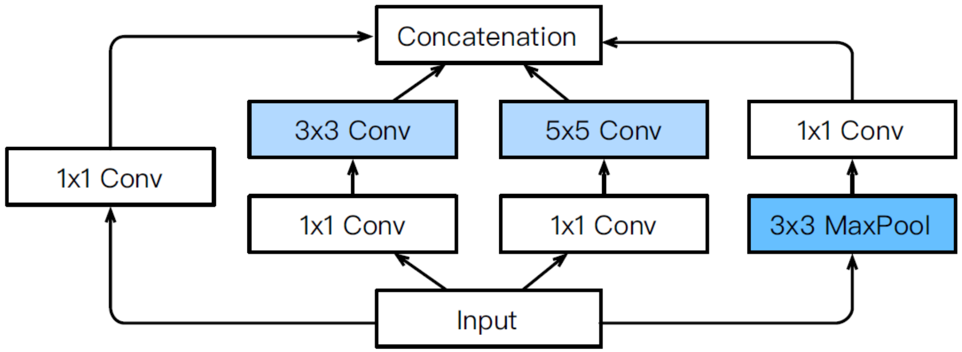
Inception V3 block for stage 3

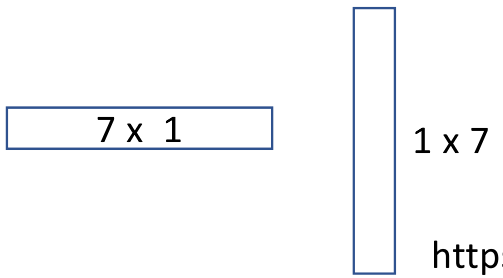


https://d2l.ai/

Inception V3 block for stage 4

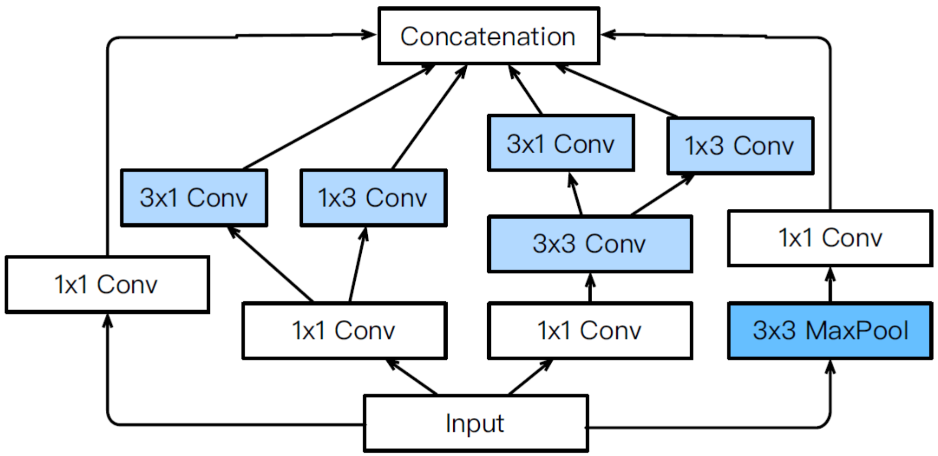


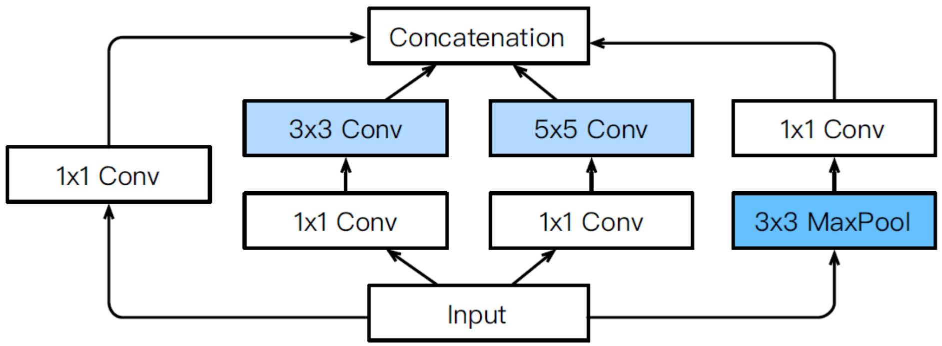




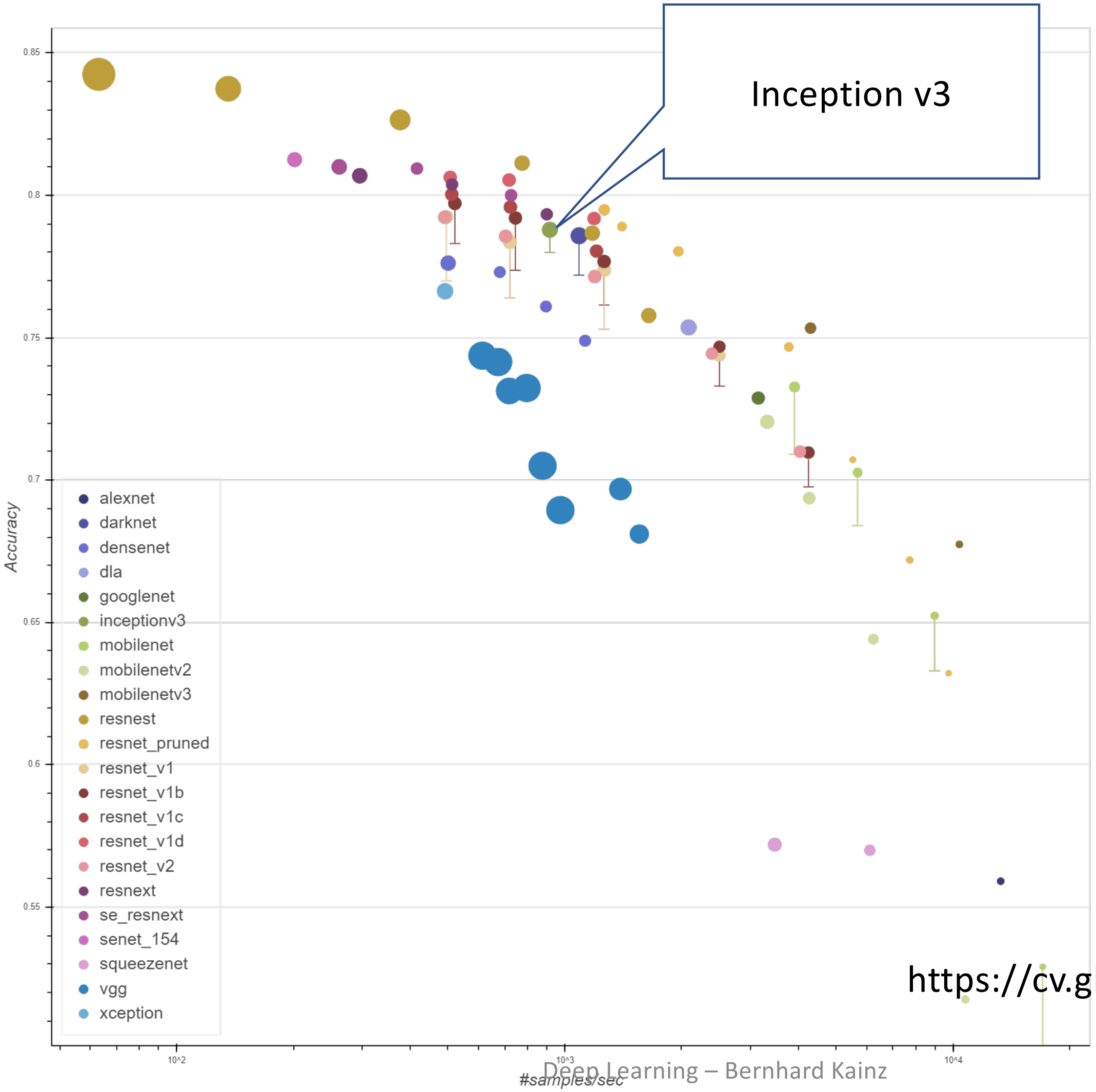
https://d2l.ai/

Inception V3 block for stage 5





https://d2l.ai/



https://cv.gluon.ai/model_zoo/classification.html

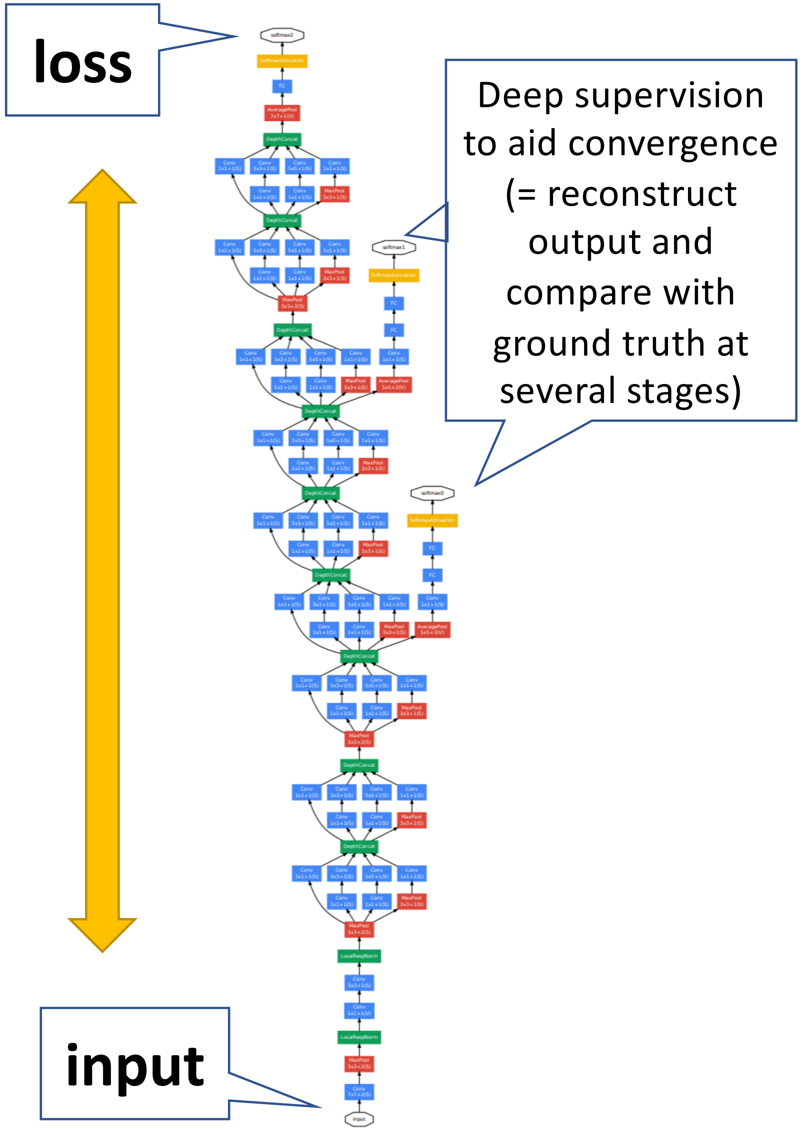
What do we learn from that?

- Dense layers are computationally and memory intensive. Real-world problems with big input tensors and many classes will prohibit their use.
- Again: 1x1 convolutions act like a multi-layer perceptron per pixel.
- Scientists are humans and need a while to understand the power of new approaches. Eventually they do but a lot of vanity is involved in the process.
- If not sure, just take all options and let the optimization decide or even learn this through trial and error (genetic algorithm, AmoebaNet)

BatchNorm

Batch Normalization

- Loss is calculated at last layer
  - Last layers learn quickly
- Data input is at first layer
  - First layers change - everything changes
  - Last layers need to relearn many times
  - Slow convergence
- This is like covariate shift...
  Can we avoid changing last layers while learning first layers?



Batch Normalization

- Can we avoid changing last layers while learning first layers?
- Fix mean and variance

$$\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \ and \ \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \varepsilon$$

and adjust it separately

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

mean

variance

Batch Normalization

- Doesn't really reduce covariate shift (Lipton et al. 2018) https://arxiv.org/abs/1805.10694
- Regularization by noise injection

Random offset

$$x_i = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

learned

learned

Random scale

  - Random shift per mini batch
  - Random scale per mini batch
- No need to add dropout (both are capacity control)
- Ideal mini batch size: 64-256

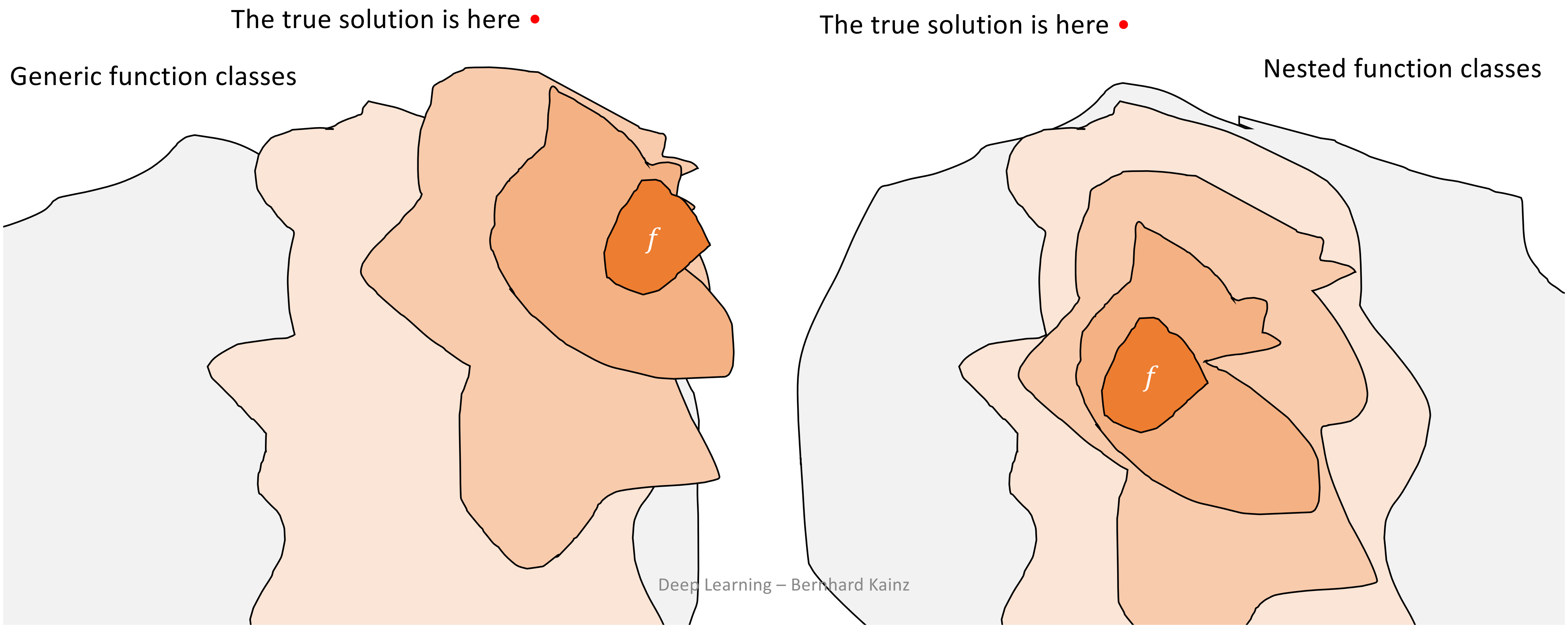
Batch Normalization

- Dense layer: One normalization for all
- Convolutional layer: One normalization per channel
- Compute **new mean and variance** for every minibatch
  - Acts as regularisation
  - Be careful when scaling up to multi-GPU training

https://xkcd.com/

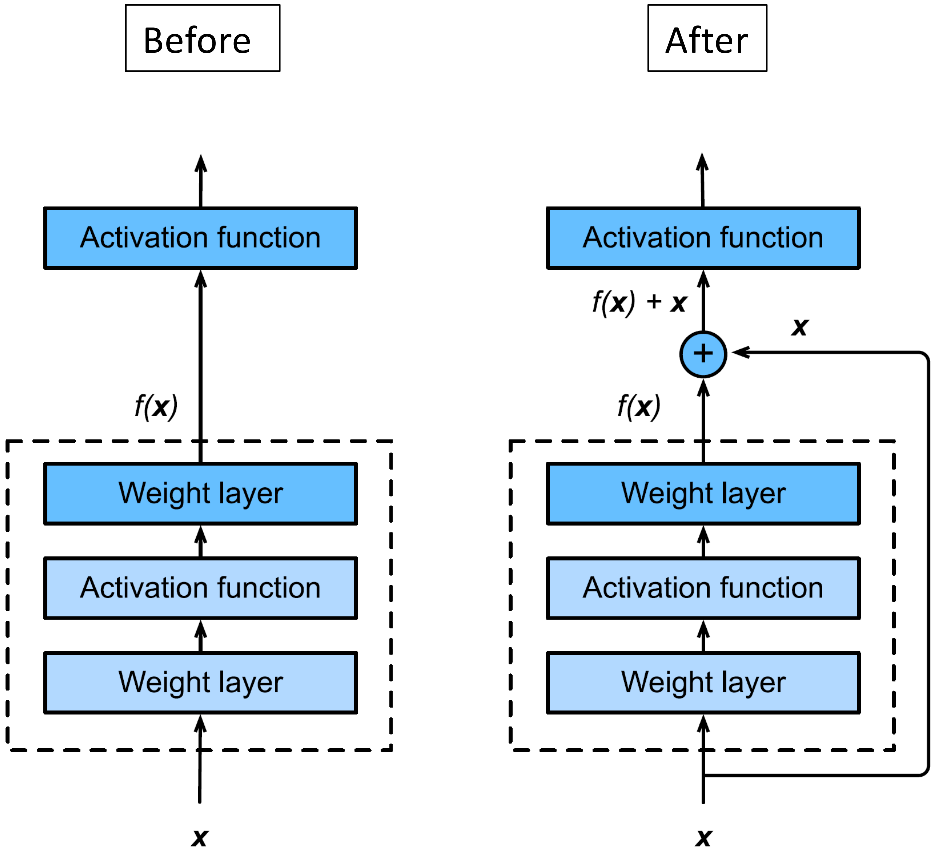
ResNet

Does adding layers improve accuracy?



Residual Networks

- Adding a layer changes function class
- We want to add to the function class
- 'Taylor expansion' style parametrization

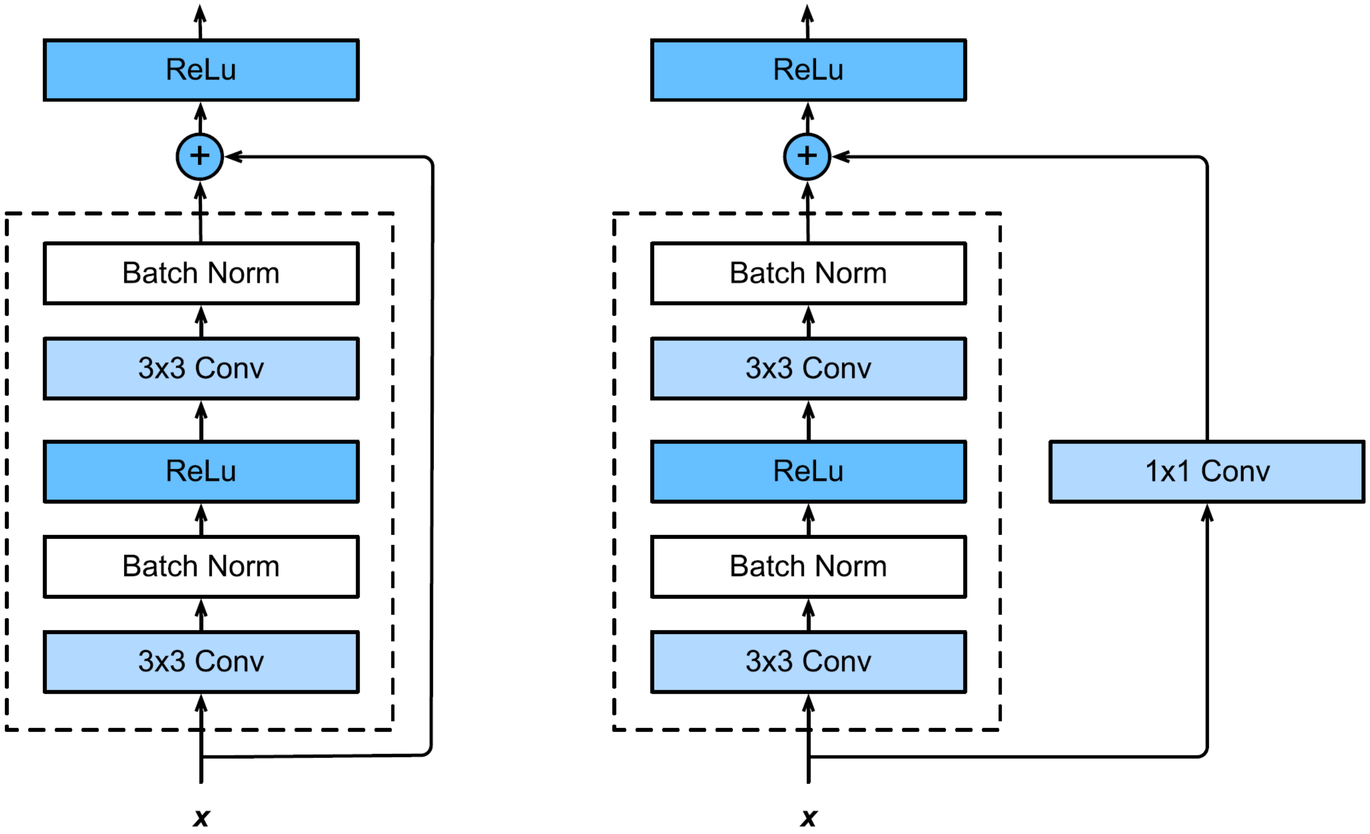


$f(x)$

He et al. 2015 https://arxiv.org/abs/1512.03385

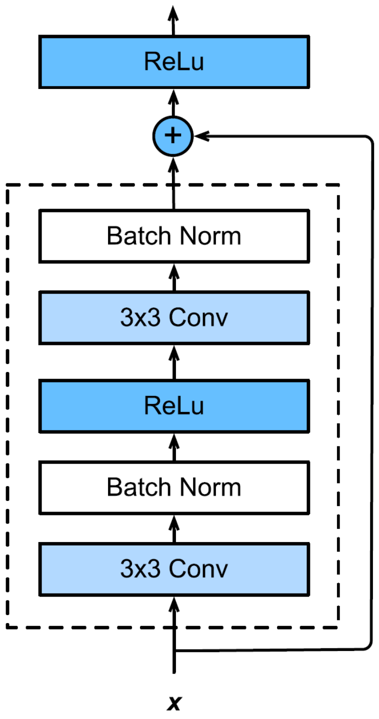
https://d2l.ai/

ResNet Block



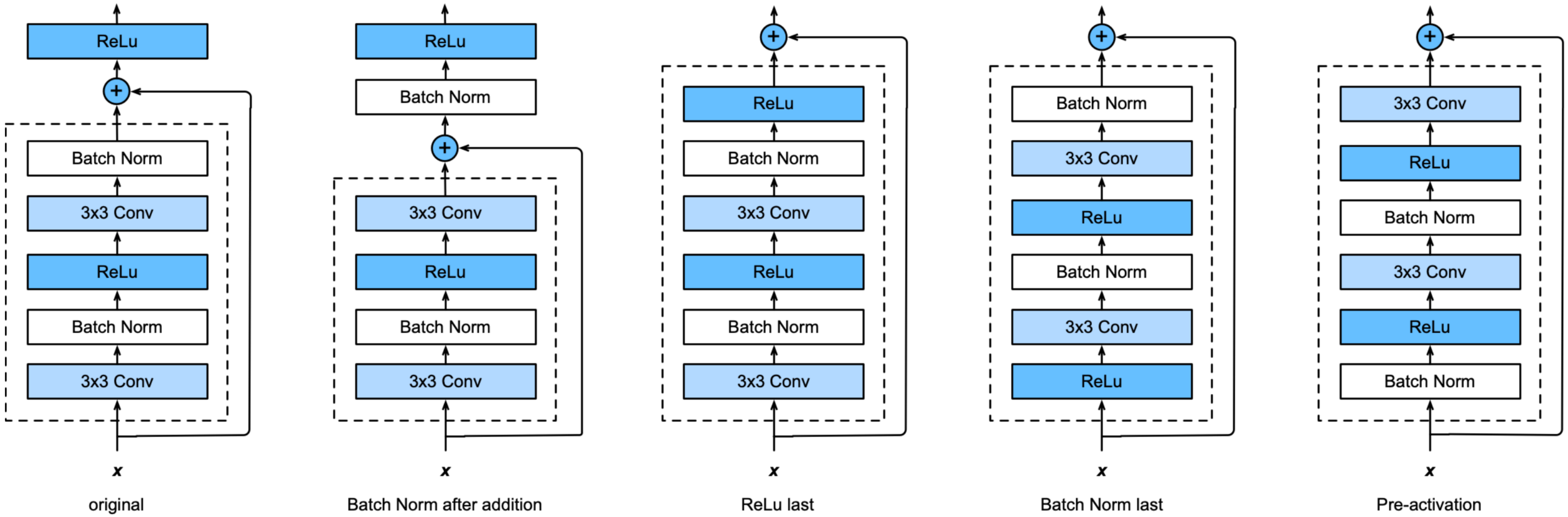
https://d2l.ai/

ResNet Block




```
def forward(self, X):
    Y = self.bn1(self.conv1(X))
    Y = nd.relu(Y)
    Y = self.bn2(self.conv2(Y))
    if self.conv3:
        X = self.conv3(X)
    return nd.relu(Y + X)
```

ResNet block flavours

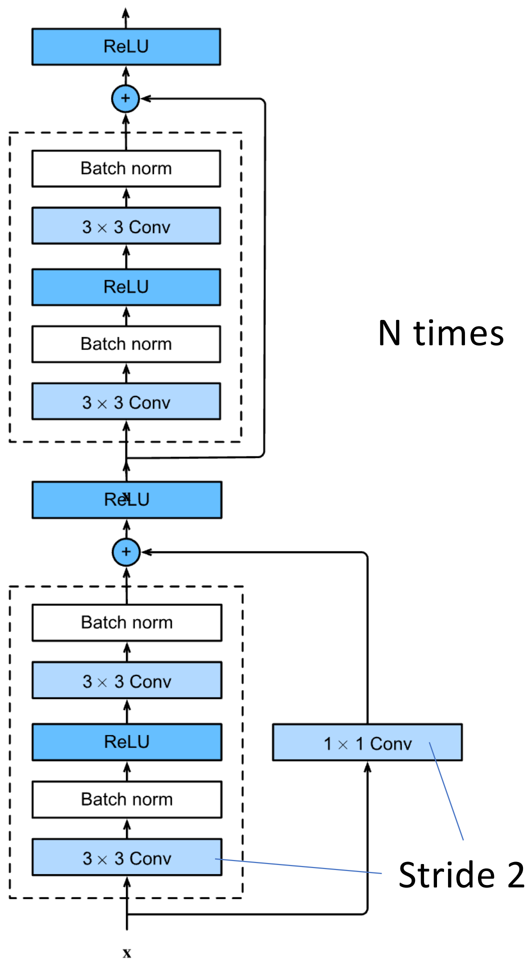


Trial and error of every permutation

https://d2l.ai/

ResNet Module

- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks



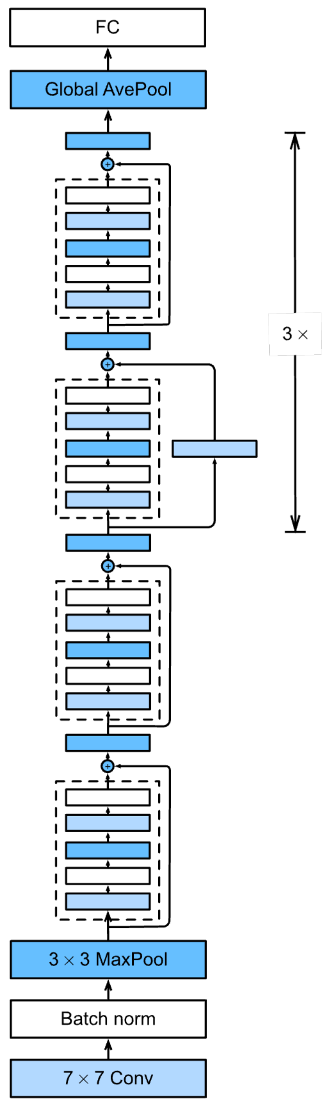
https://d2l.ai/

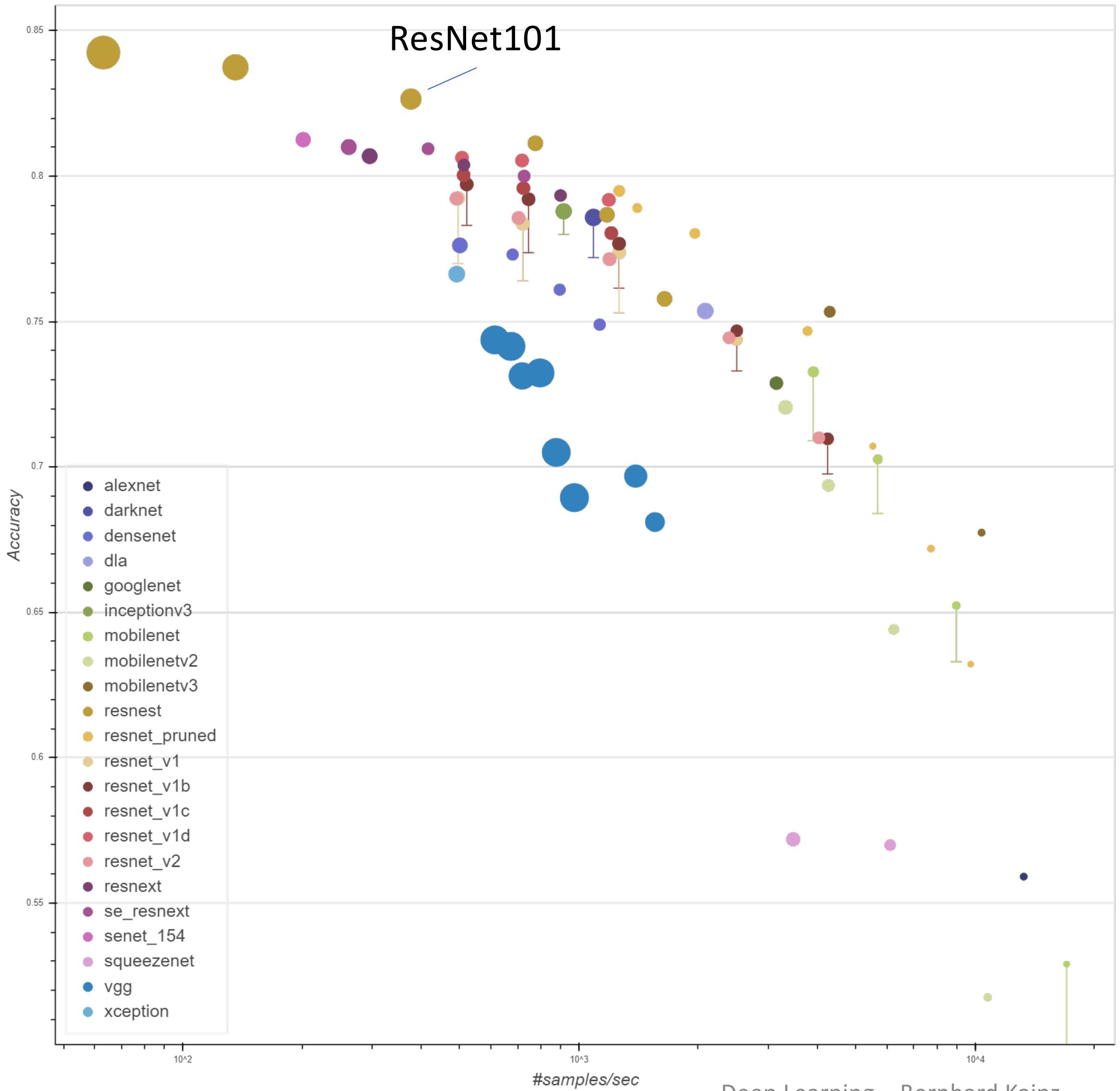
ResNet

Same block structure as e.g. VGG or GoogleNet

- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

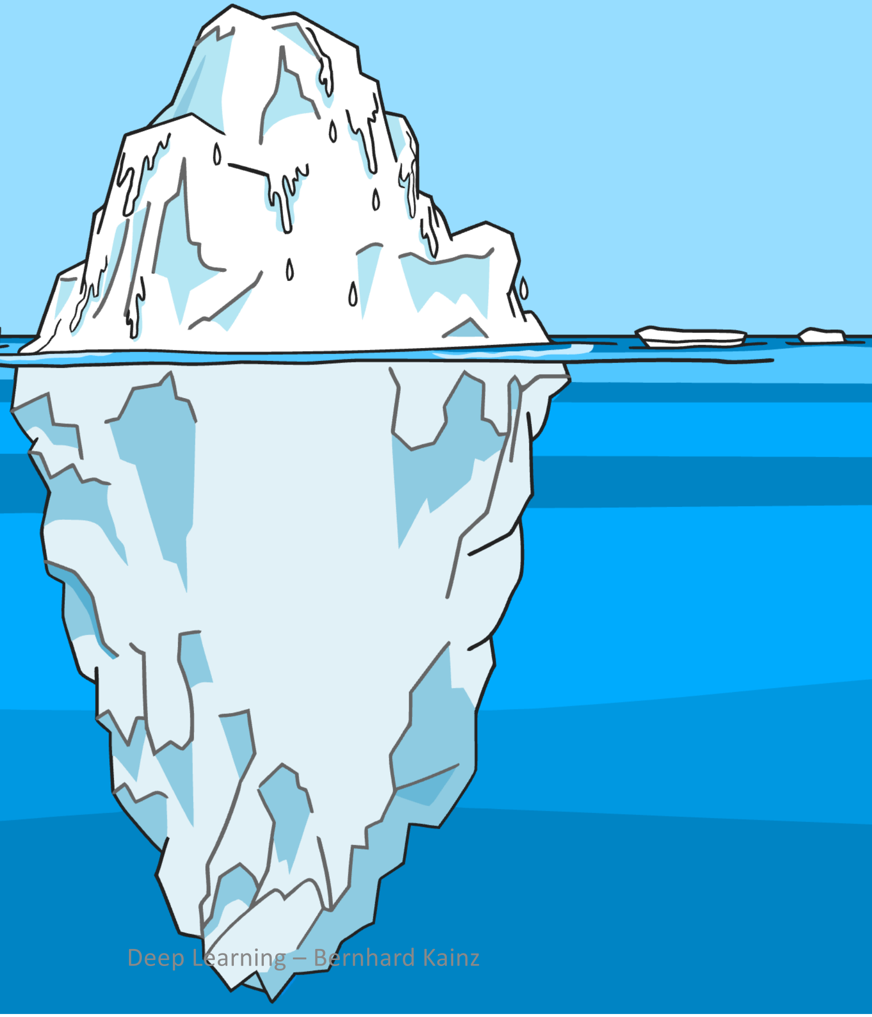
- Trainable at scale
- Variant name depends on how many blocks (18 layers = ResNet-18 ->)



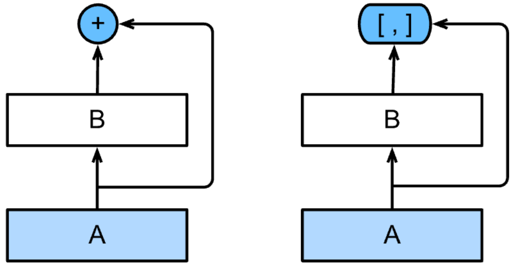


https://cv.gluon.ai/model_zoo/classification.html

More ideas…



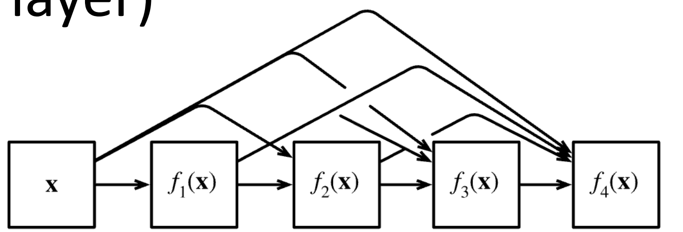
DenseNet



- Huang et al., 2016 https://arxiv.org/abs/1608.06993
- ResNet combines x and $f(x)$
- DenseNet uses higher order 'Taylor series' expansion

$$
\begin{aligned}
x_{i+1} &= [x_i, f_i(x_i)] \\
x_1 &= x \\
x_2 &= [x, f_1(x)] \\
x_3 &= [x,\ f_1(x), f_2([x, f_1(x)])]
\end{aligned}
$$

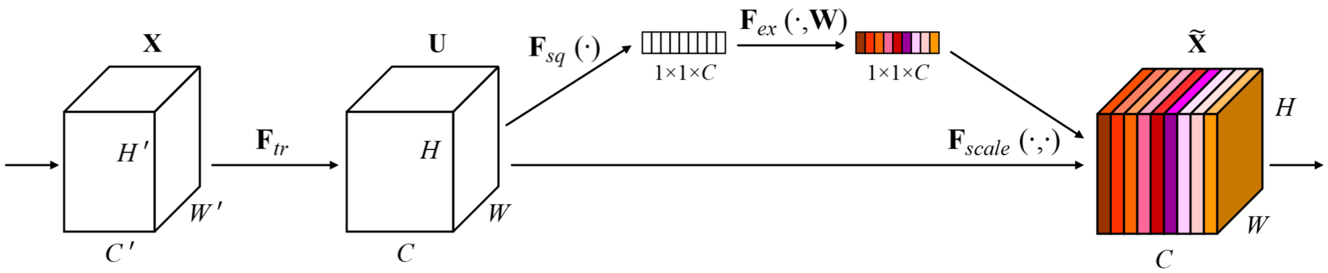
- Occasionally need to reduce resolution (transition layer)



https://d2l.ai/

Squeeze-Excite Net

- Hu et al., 2017 https://arxiv.org/abs/1709.01507



- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

Things to explore

- AutoML (find best model architecture automatically Google Cloud AutoML)
- Hypernetworks (a network that proposes the weights for another network), also neural processes
- Networks with memory, e.g. kanerva machine
- Almost no new basic architectures accepted nowadays (see https://nips.cc/virtual/2020/public/cal_main.html NeurIPS 2020 programme, focuses on meta findings)
- Attention! (second part of the course)

Summary

- **Inception**
  - Inhomogeneous mix of convolutions (varying depth)
  - Batch norm regularization
- **ResNet**
  - Taylor expansion of functions
  - ResNext decomposes convolutions
- **Model Zoo**
  - DenseNet, ShuffleNet, Separable Convolutions, …

https://in.pinterest.com/pin/556124253981912489/

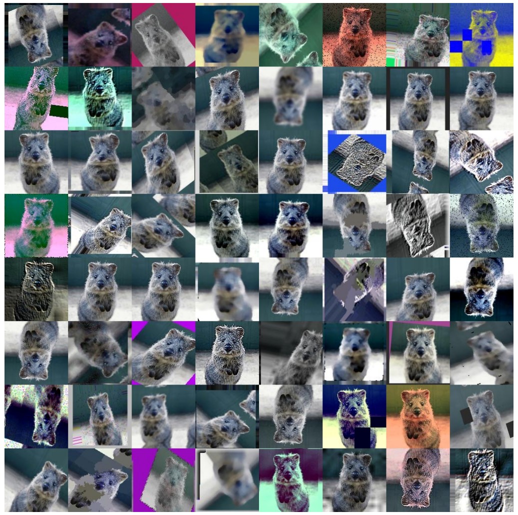
What do we learn from that

- Deeper is not necessarily better if the function space is not regularised
- ResNet is the workhorse of Deep learning (for now. Do you have a better idea that hasn't been tried yet? Let me know but look on arXiv first!)
- Lot's of variations have been proposed but it often boils down to how you train a network and for what purpose.

Data Augmentation

Input augmentation

- Artificially inflate training data size through applying expected transformations during training
- https://github.com/aleju/imgaug
- https://pytorch.org/docs/stable/torchvision/transforms.html
- Excellent regularizer against overfitting



Transformations

- Random
  - flipping
  - scaling
  - rotations
  - intensity/contrast variations
  - cropping/padding
  - noise
  - affine transformations
  - perspective transformations