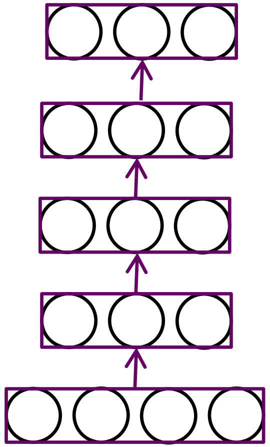
Pretrained Models and Transfer learning

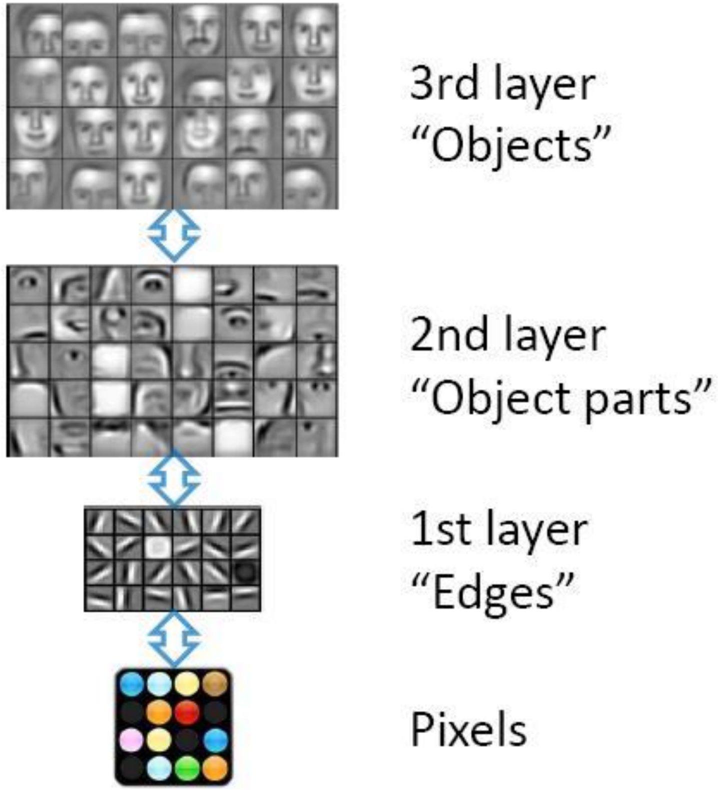
Shailesh S

Why Stop At One Hidden Layer?

E.g., vision hierarchy for recognizing handprinted text

Word	output layer
Character	hidden layer 3
Stroke	hidden layer 2
Edge	hidden layer 1
Pixel	input layer




3rd layer
“Objects”
2nd layer
“Object parts”
1st layer
“Edges”
Pixels

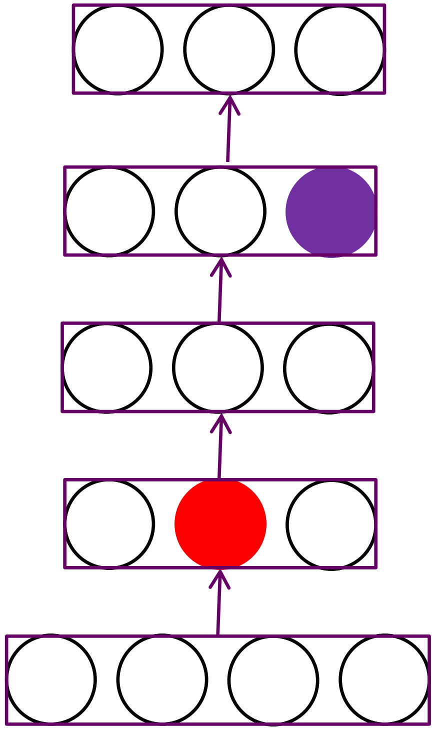
Challenges in Deep Learning

- Requires Large dataset for Training
- Overfitting or lack of generalization
- Vanishing or Exploding Gradient Problem
- Appropriate Learning rate
- Covariate Shift
- Effective Training

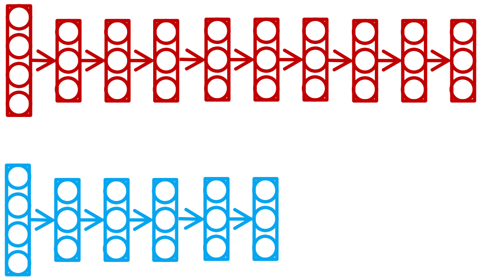
Why Deeply Layered Networks Fail

Credit assignment problem

- How is a neuron in layer 2 supposed to know what it should output until all the neurons above it do something sensible?
- How is a neuron in layer 4 supposed to know what it should output until all the neurons below it do something sensible?



Deeper Vs. Shallower Nets



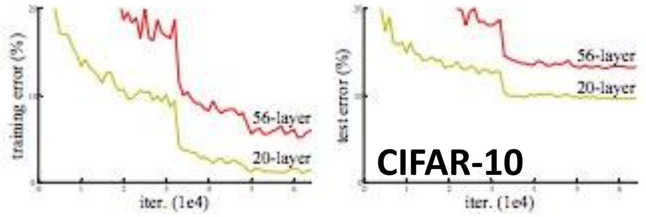
Deeper net can represent any mapping that shallower net can

- Use identity mappings for the additional layers

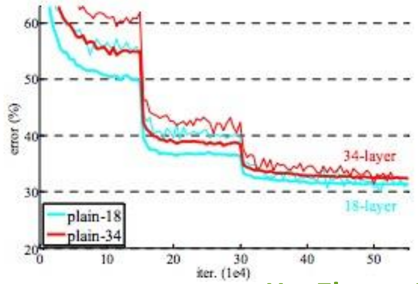
Deeper net in principle is more likely to overfit

But in practice it often underfits *on the training set*

- Degradation due to harder credit-assignment problem
- Deeper isn't always better!



CIFAR-10



ImageNet

thin=train, thick=valid.

He, Zhang, Ren, and Sun (2015)

Why Deeply Layered Networks Fail

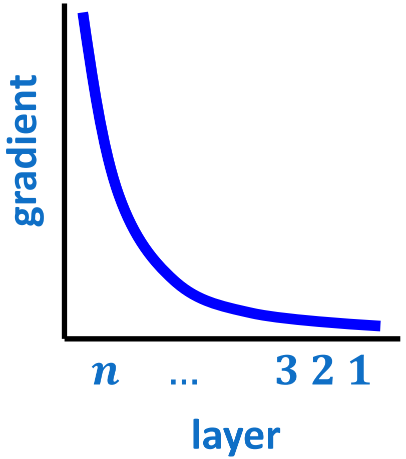
Vanishing gradient problem

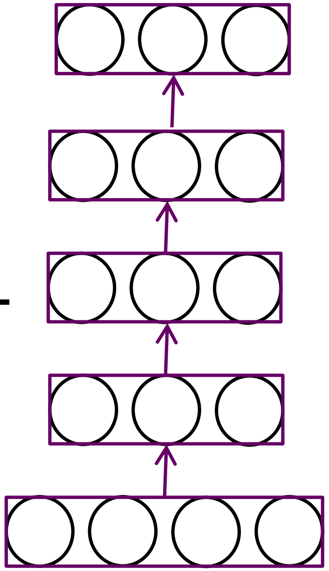
- With logistic or tanh units

$$y_j = \frac{1}{1 + \exp(-z_j)} \qquad \frac{\partial y_j}{\partial z_j} = y_j(1 - y_j)$$

$$y_j = \tanh(z_j) \qquad \frac{\partial y_j}{\partial z_j} = (1 + y_j)(1 - y_j)$$

- Error gradients get squashed as they are passed back through a deep network



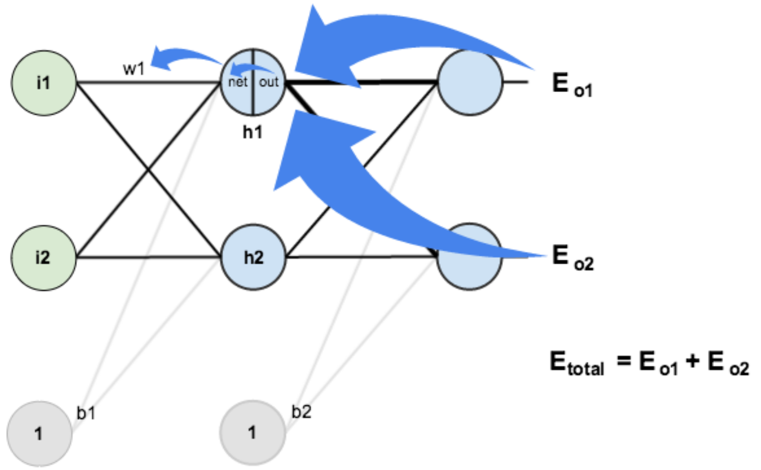


Why Deeply Layered Networks Fail

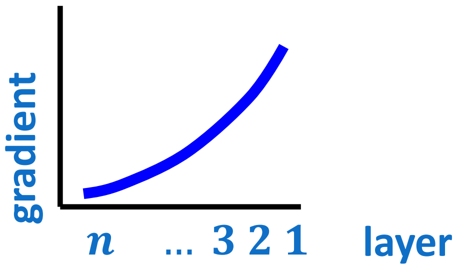
$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



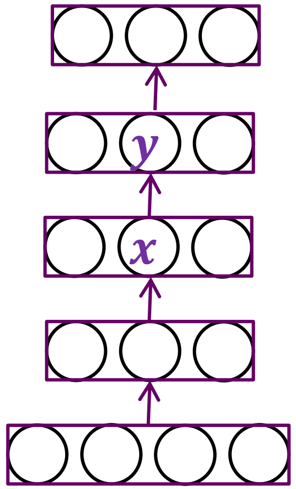
Why Deeply Layered Networks Fail



$$y = \max(\mathbf{0}, z)$$

$$\frac{\partial y}{\partial z} = \begin{cases} 0 & z \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

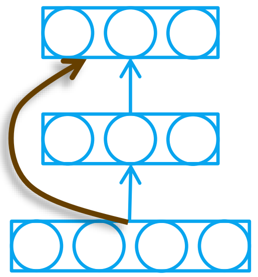
$$\frac{\partial y}{\partial x} = \begin{cases} 0 & y = 0 \\ w_{yx} & \text{otherwise} \end{cases}$$



Exploding gradient problem

- with linear or ReLU units
- can be a problem when $\left|\sum_y \frac{\partial y}{\partial x}\right| > 1$

Trick From Long Ago To Avoid Local Optima



Add direct connections from input to output layer

Easy bits of the mapping are learned by the direct connections

- Easy bit = linear or saturating-linear functions of the input
- Often captures 80-90% of the variance in the outputs

Hidden units are reserved for learning the hard bits of the mapping

- They don’t need to learn to copy activations forward for the linear portion of the mapping

Problem

- Adds a lot of free parameters

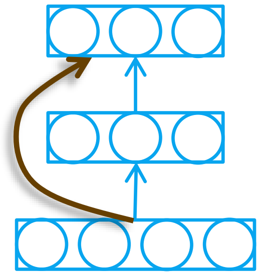
Latest Tricks

Novel architectures that

- skip layers
- have linear connectivity between layers

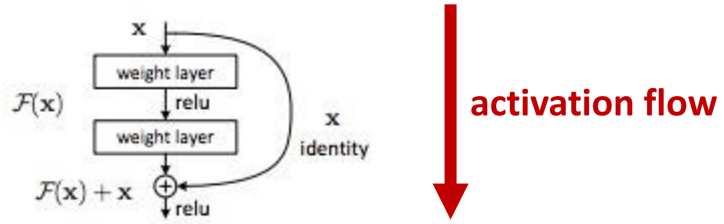
Advantage over direct-connection architectures

- no/few additional free parameters



Deep Residual Networks (ResNet)

Add linear short-cut connections to architecture

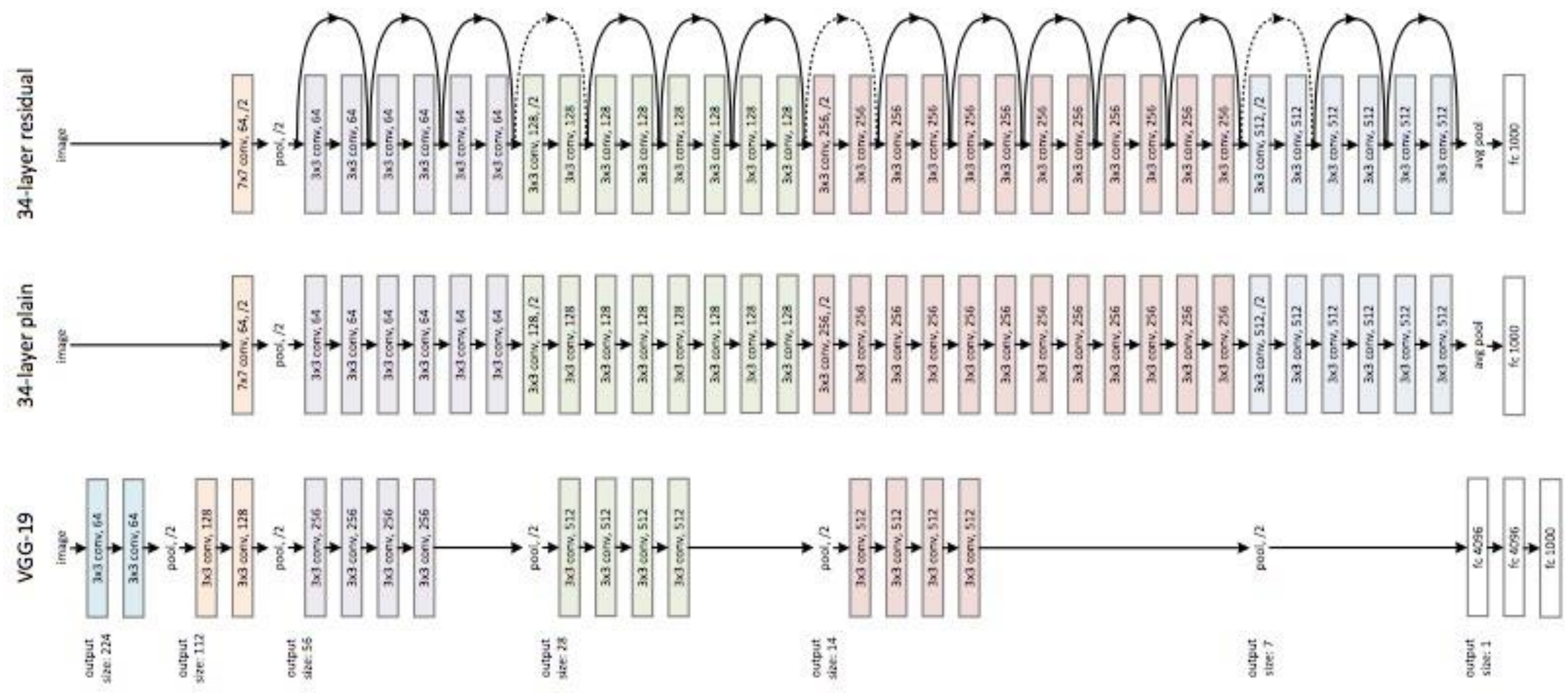

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
activation flow

Basic building block

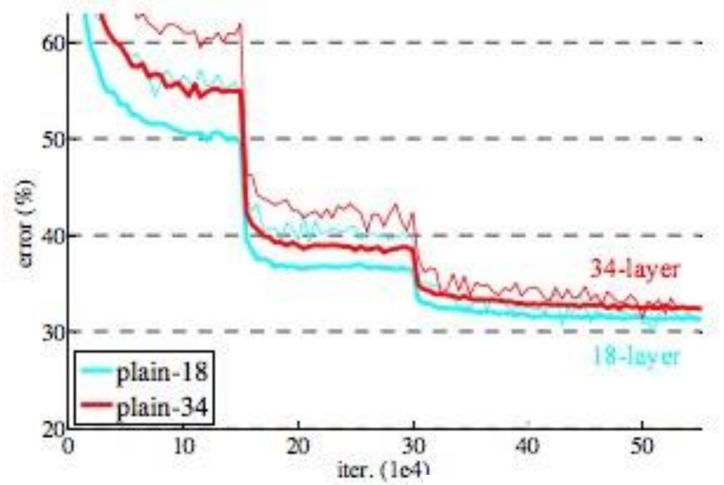
- $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{W}_2 \boldsymbol{\sigma}(\boldsymbol{W}_1 \boldsymbol{x})$
- $\boldsymbol{y} = \boldsymbol{\sigma}(\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x})$

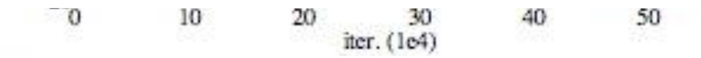
Variations

- allow different input and output dimensionalities: $\boldsymbol{y} = \boldsymbol{\sigma}(\mathcal{F}(\boldsymbol{x}) + \boldsymbol{W}_3 \boldsymbol{x})$
- $\mathcal{F}(\boldsymbol{x})$ can have an arbitrary number of layers

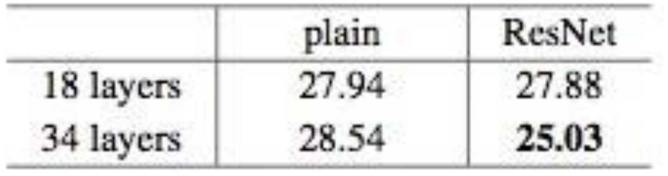

34-layer residual
34-layer plain
VGG-19
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
fc 4096
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: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

ResNet





Top-1 Error %



	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

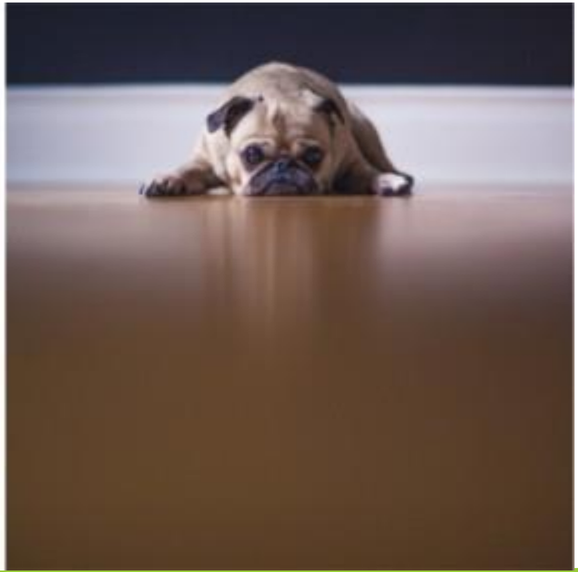
Inception Networks

Inception

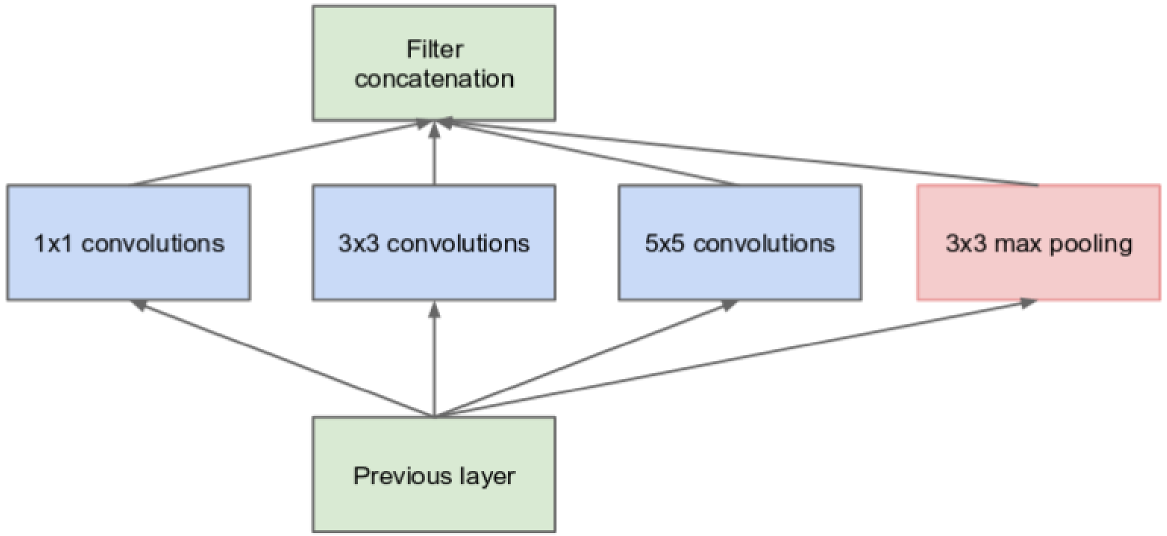
- traditional deep neural networks, which is well known for stacked convolutions
- Inception networks are well known for parallel stacked convolutions
- Inception block contains different size of convolutions and one max pooling layers which is arranged in a parallel manner.It is aggregated for the final output for the inception block
- I.e, Computing 1*1,3*3, 5*5 convolutions with in the same module of the network which is arranged in a parallel manner

Inception

- In every image, there is a huge variation in the location of information
- Filter size should be chosen according to the information content



Google Net



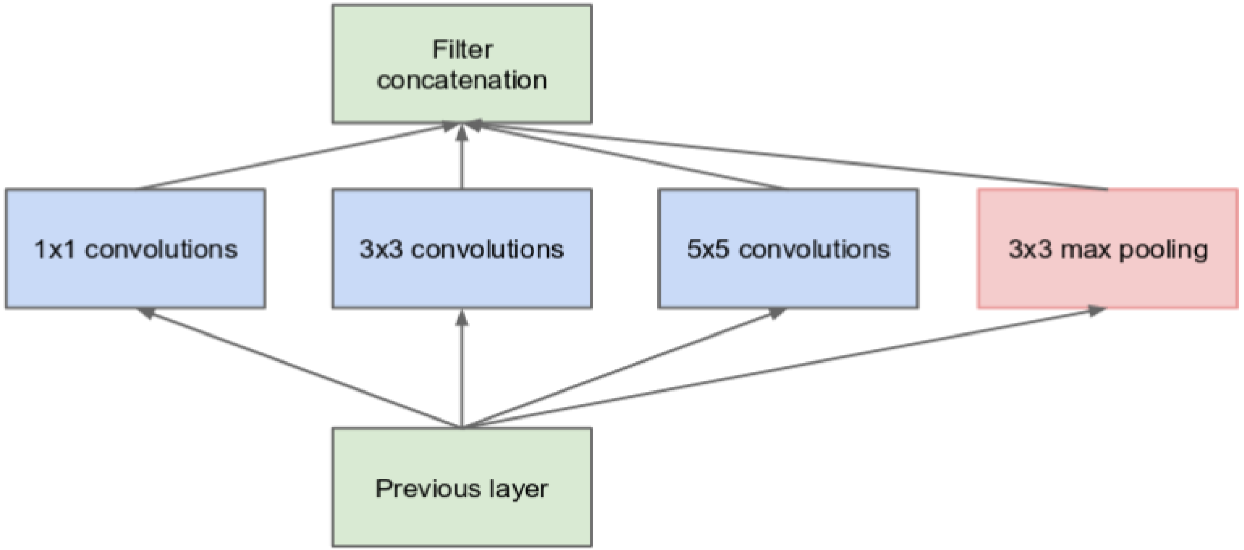
(a) Inception module, naïve version

Architecture based on Inception network

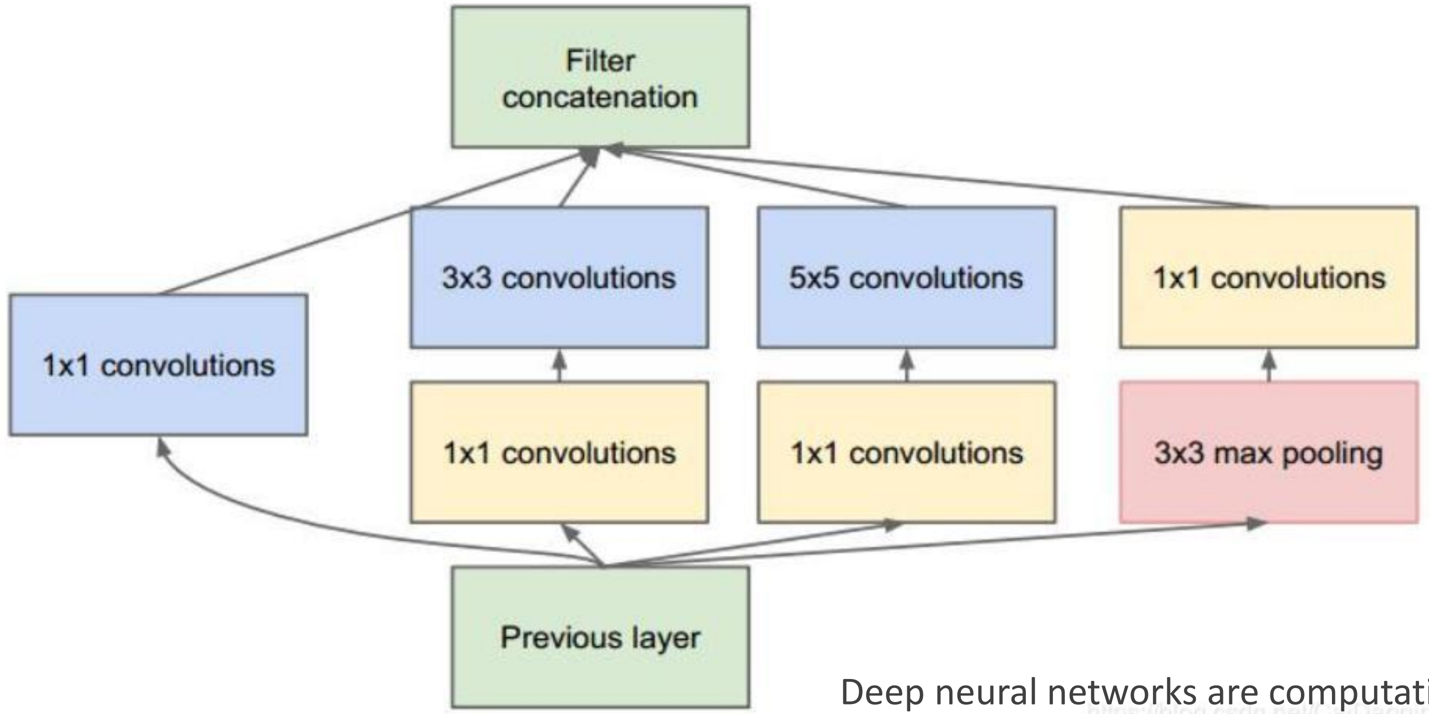
- Inception v1(2014)
- Inception v3(2015)
- Inception v3(2015)
- Inception v4(2016)
- Inception ResNet(2016)

Adaptability of the different scales can be accommodated

Overfitting can be prevented



(a) Inception module, naïve version



Deep neural networks are computationally expensive. To make it cheaper, the author limit the number of input channels by adding an extra 1*1 convolution before 5*5 and 3*3 convolutions

Transfer Learning

Transfer Learning

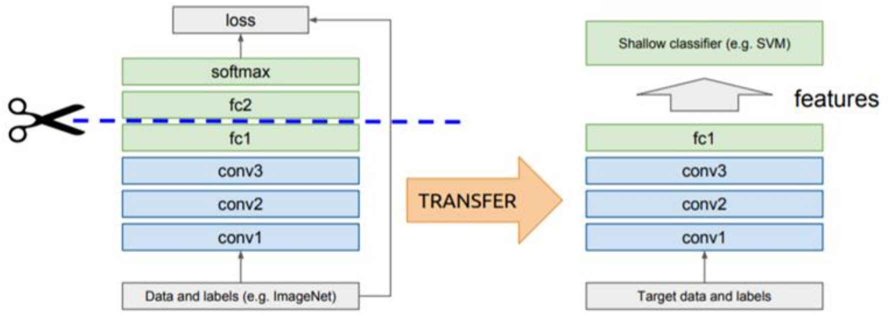
- Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.
- The weights in re-used layers may be used as the starting point for the training process and adapted in response to the new problem.
- This usage treats transfer learning as a type of weight initialization scheme.
- This may be useful when the first related problem has a lot more labeled data than the problem of interest and the similarity in the structure of the problem may be useful in both contexts.
- **Transfer learning** consists of taking features learned on one problem, and leveraging them on a new, similar problem.
- Taking model learned on one problem , and update it for the new ,similar problem

Feature Extraction

The pre-trained model can be used as a separate feature extraction program, in which case input can be pre-processed by the model or portion of the model to a given an output (e.g. vector of numbers) for each input image, that can then use as input when training a new model.

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



Freezing layers:

- Take layers from previously trained model
- Freezing them so as to avoid destroying any of the information during future training rounds
- Add some new trainable layers on top of the frozen layers
- Train the new layers on your dataset

Fine-tuning: supervised domain adaptation

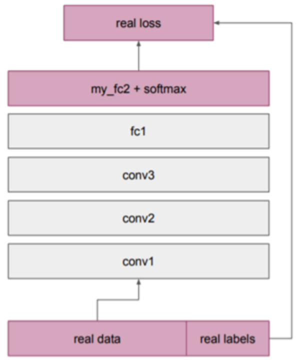
Train deep net on “nearby” task for which it is easy to get labels using standard backprop

- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

Cut off top layer(s) of network and replace with supervised objective for target domain

Fine-tune network using backprop with labels for target domain until validation loss starts to increase

Aligns D_S with D_T



Freeze or fine-tune?

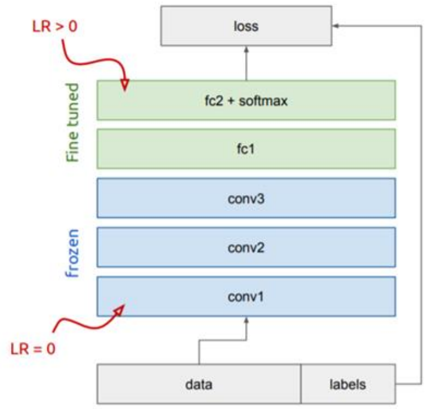
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning


LR > 0
loss
Fine tuned
fc2 + softmax
fc1
conv3
frozen
conv2
conv1
LR = 0
data
labels