Convolutional Neural Networks (CNN)

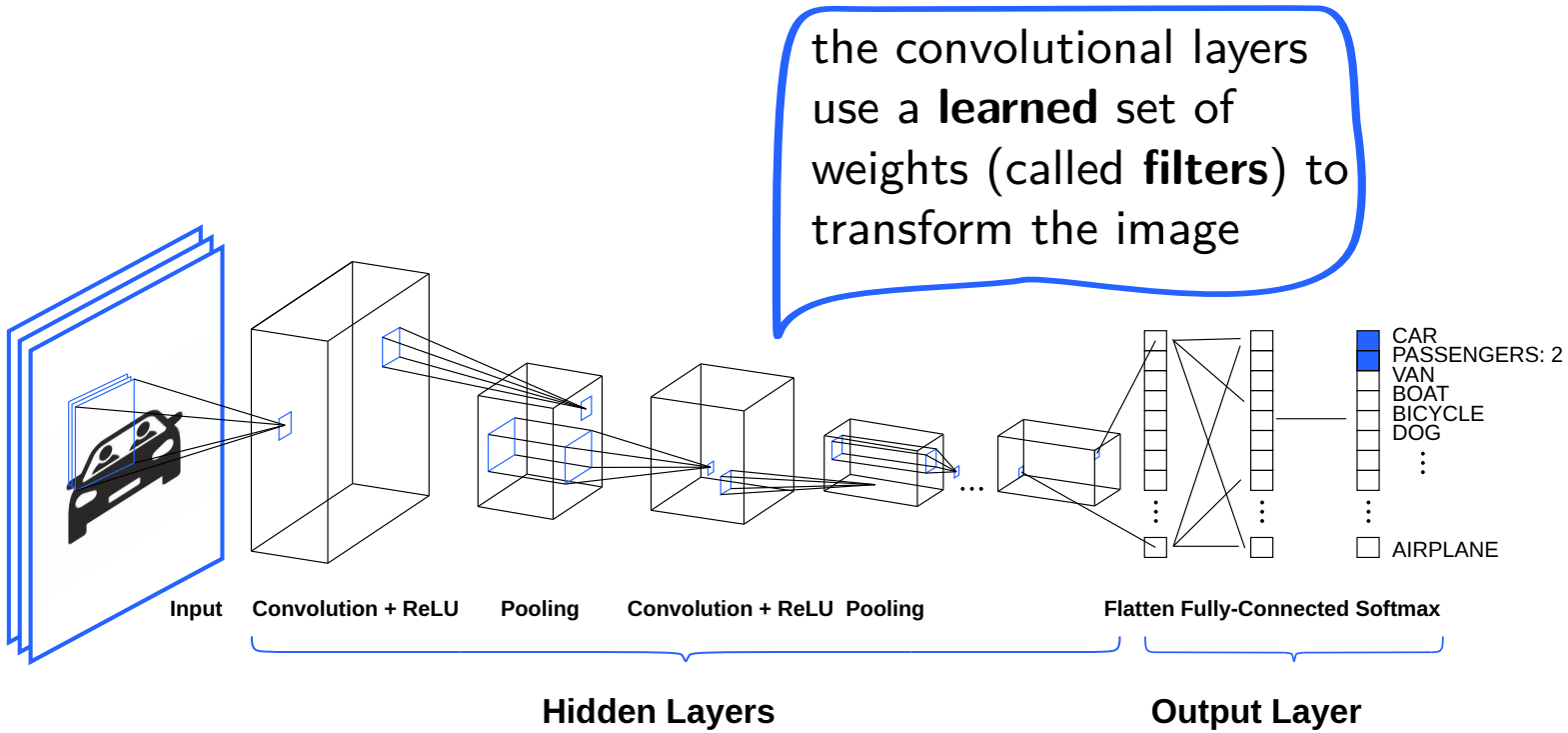
State-of-the-Art (SOTA) Architectures and Transfer Learning

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

August 12, 2023

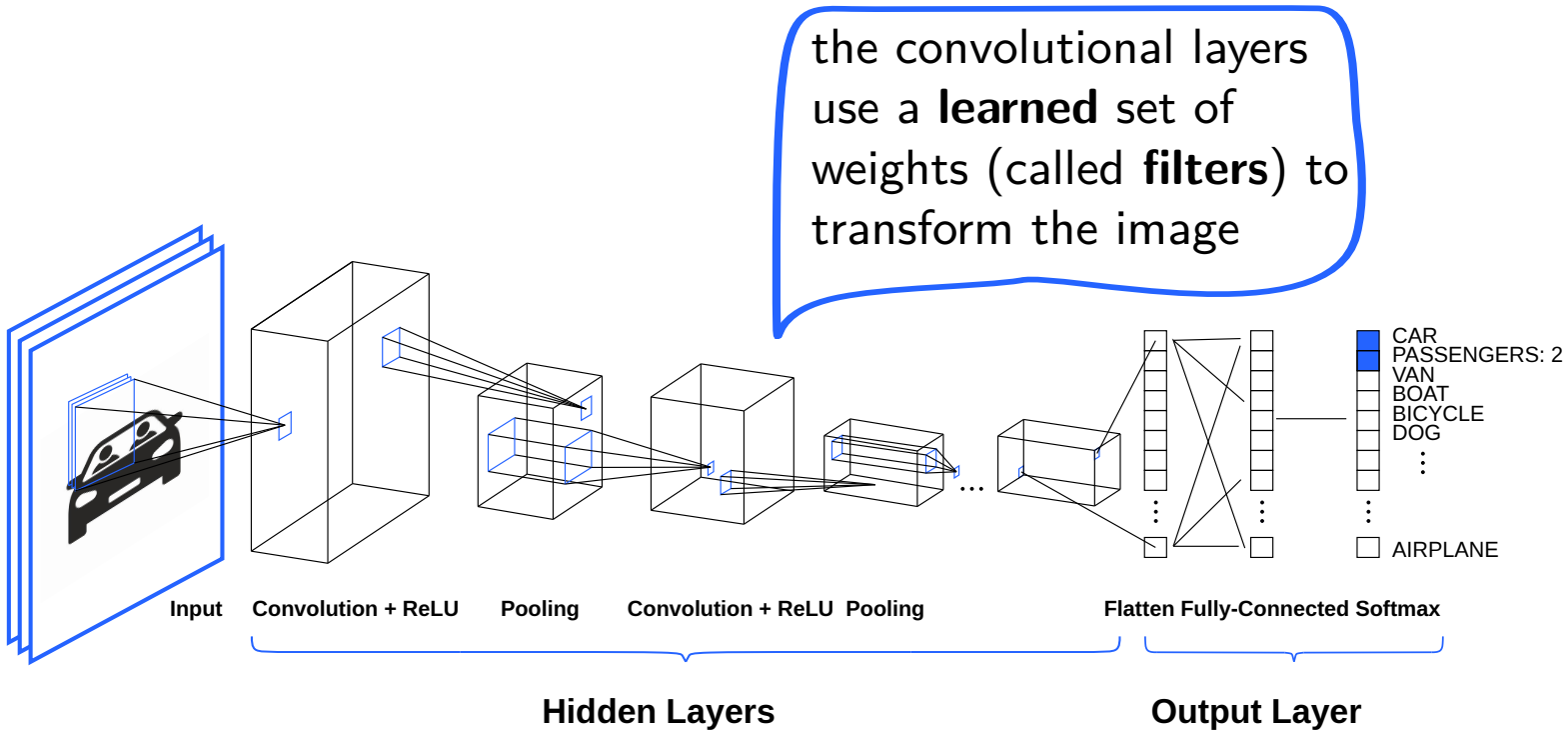
Recap: Neural Network Architectures

How to construct a CNN?



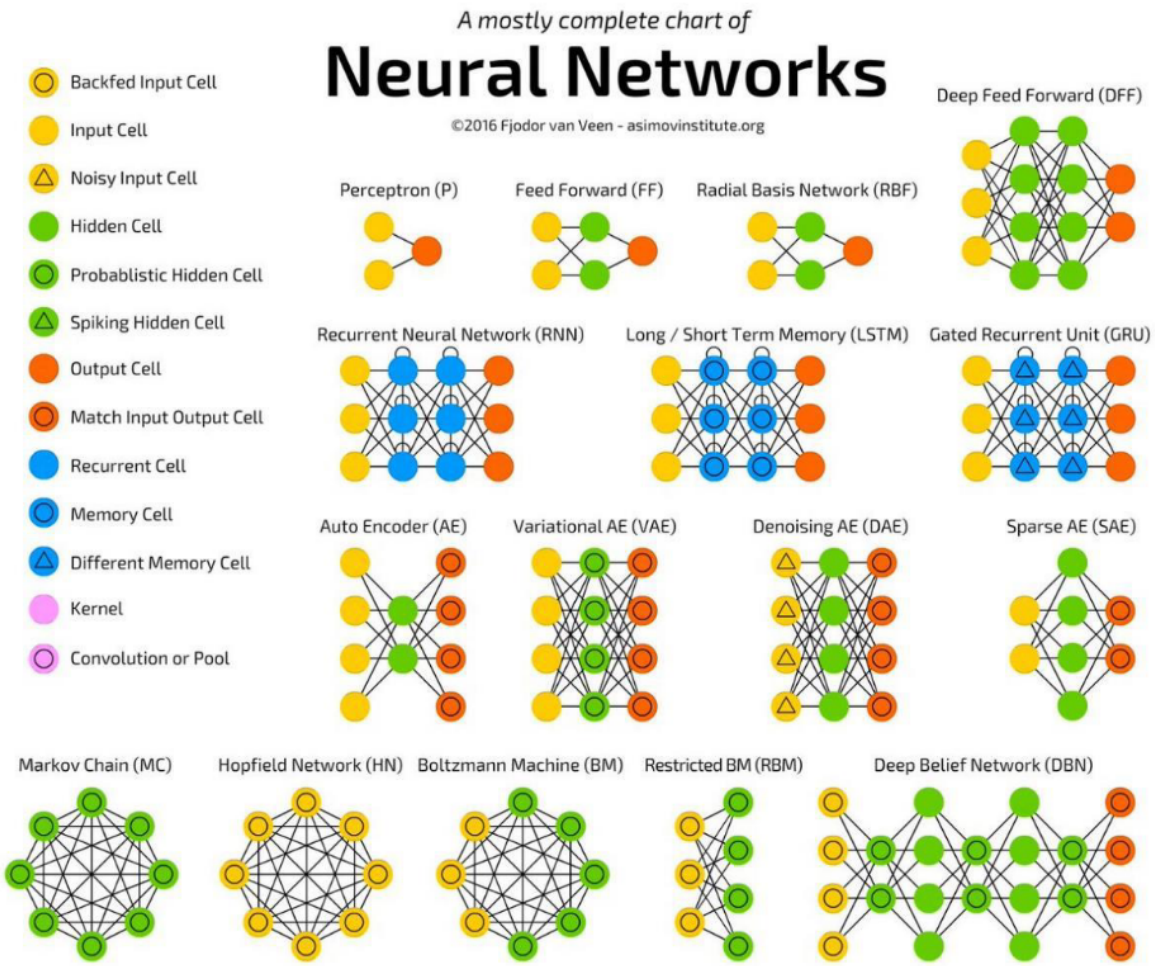
Recap: Neural Network Architectures

How to construct a CNN?



Goal: Extract features that best helps us to perform our downstream task (e.g.: object recognition, classification).

Recap: Neural Network Architectures



Sasen Cain (@spectralradius)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

a giant training (and test) set:

The **ImageNet project** is a large visual database designed for use in visual object recognition software research.

ImageNet has more than 15 million human-labeled high-resolution images with 22000 categories.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

a giant training (and test) set:

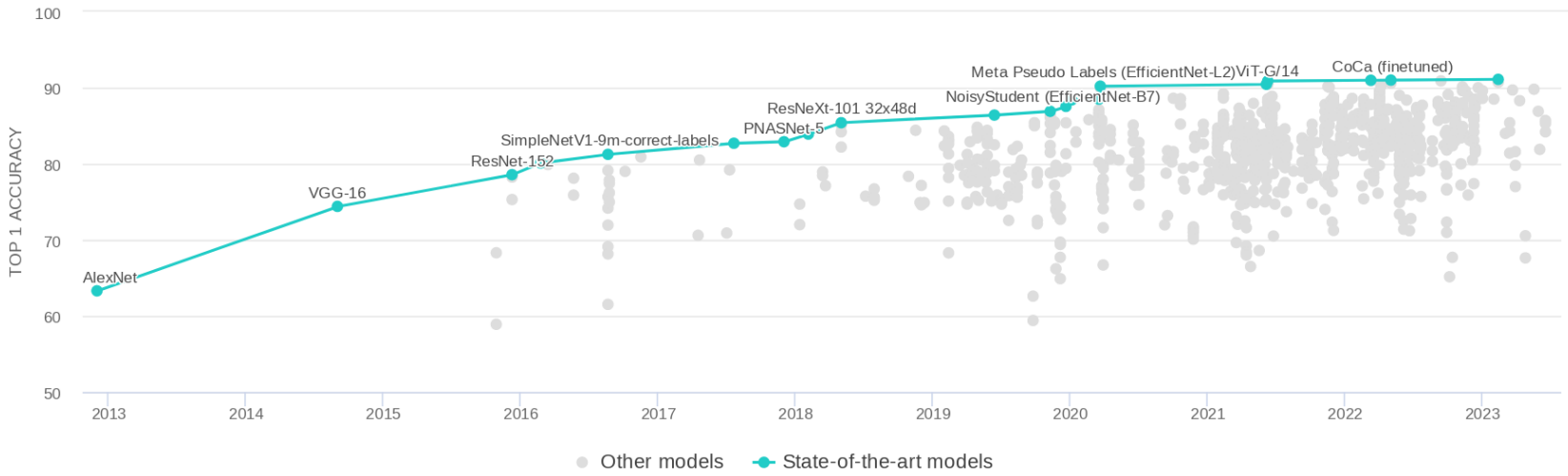
The **ImageNet project** is a large visual database designed for use in visual object recognition software research.

ImageNet has more than 15 million human-labeled high-resolution images with 22000 categories.

For at least 1 million images, also bounding boxes of objects are provided.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

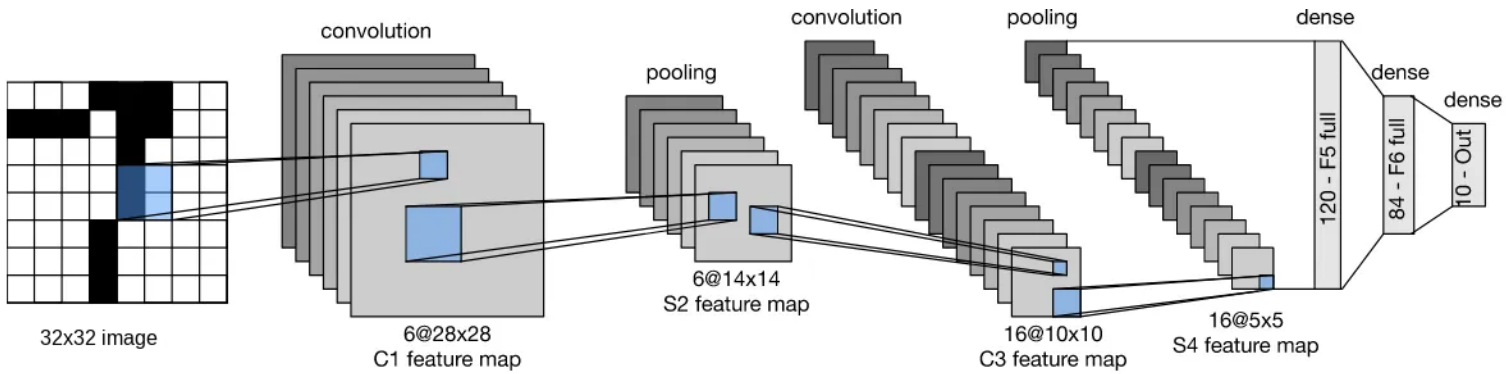
New model architectures consistently outperform even human error rate.



credit: https://paperswithcode.com/

LeNet-5

Lenet-5 was proposed by Y. LeCun (1998) in **Gradient-Based Learning Applied to Document Recognition**. This CNN is used to predict handwritten numeric characters in greyscale images.



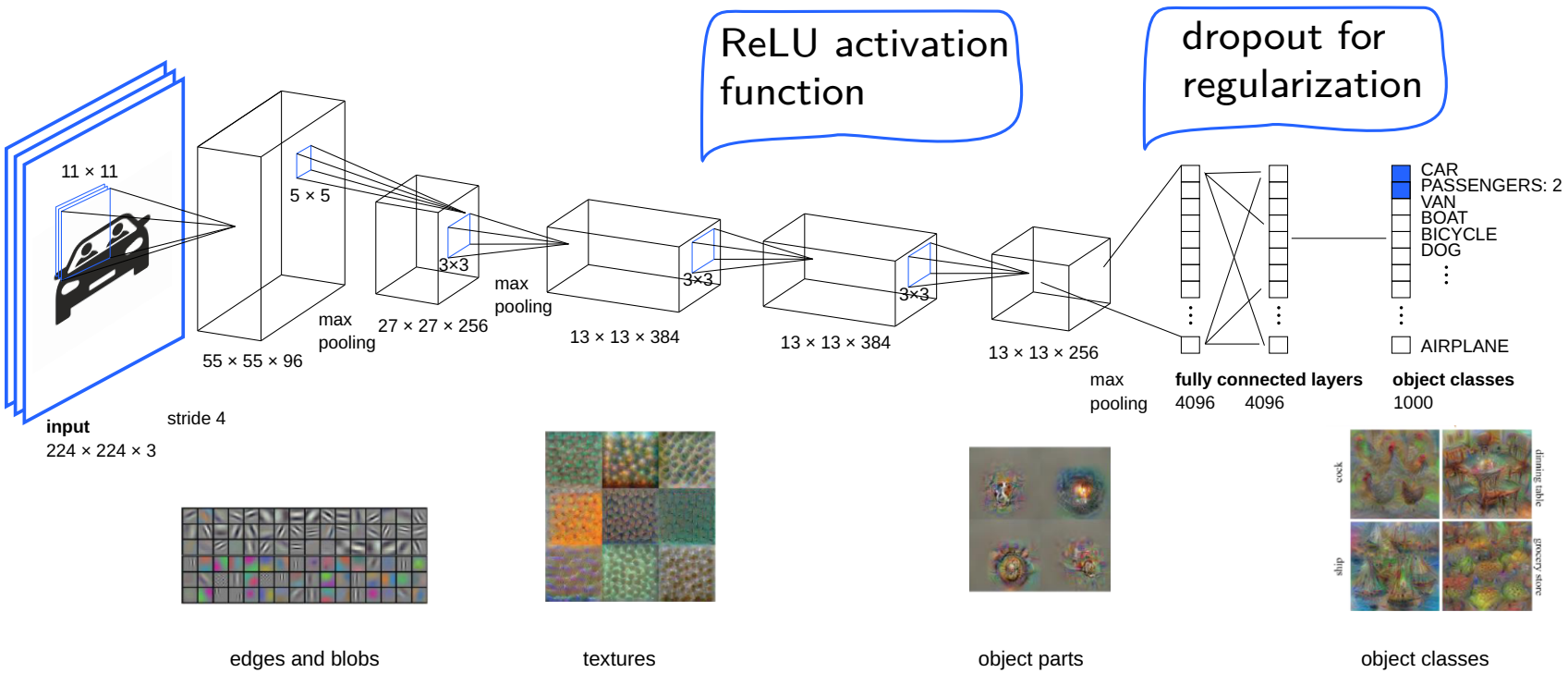
Excluding pooling, LeNet-5 consists of 5 layers:

- 2 convolution layers with kernel size 5×5, followed by
- 3 fully connected layers.

Tanh activation function is used except for the last layer (which has softmax). LeNet-5 has 60,000 trainable parameters.
Input: 32×32 pixel image. Largest character is 20×20. Pixel values are normalized (mean of pixels $= 0$, std of pixels $=1$).

AlexNet

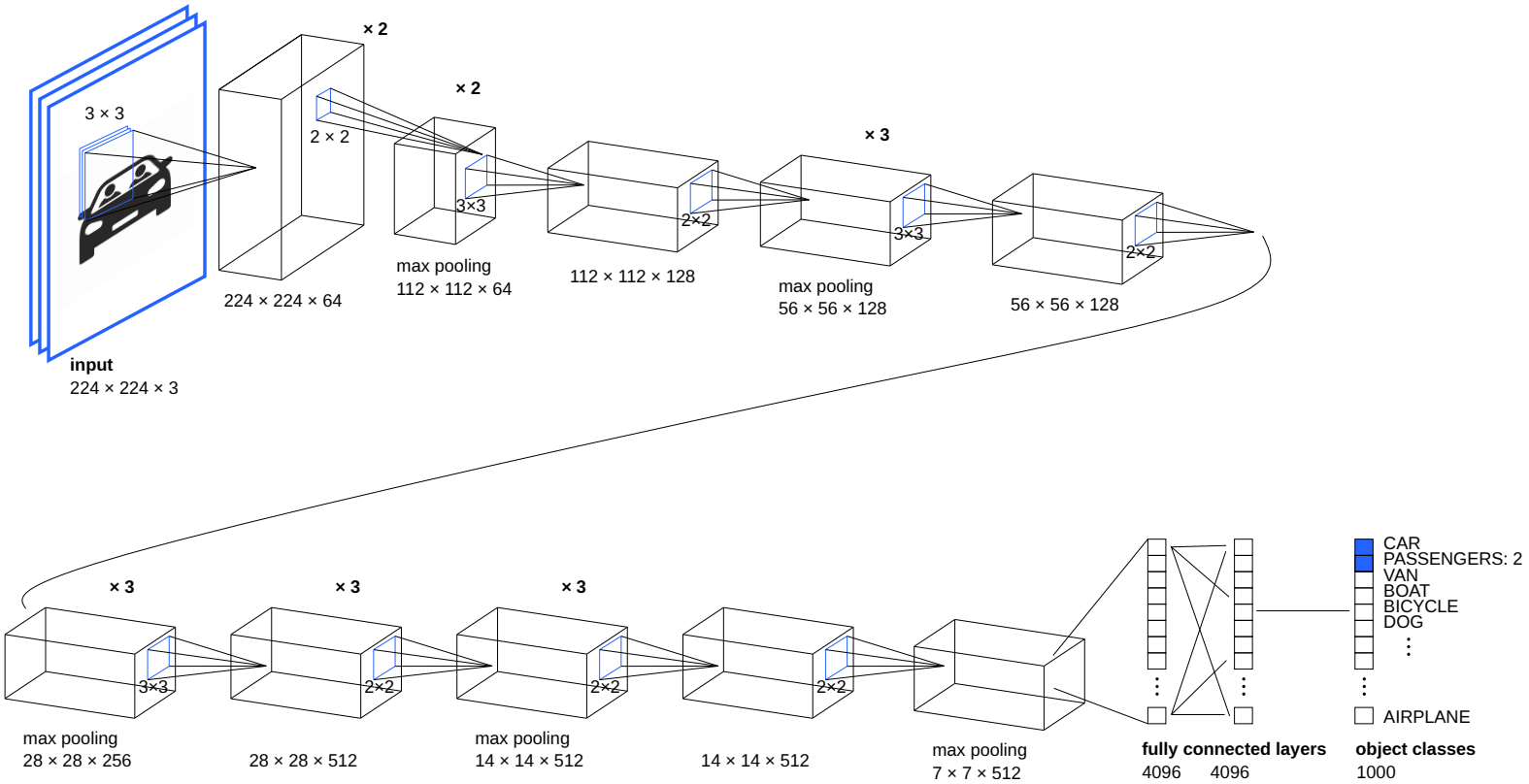
AlexNet was developed in 2012 by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton at UToronto. It was trained on 1.2M images from ImageNet with data augmentation.



In the 2012 ILSVRC, AlexNet archieved a top-5 error of 15.3%, more than 10.8 percentage points lower than the runner-up.

VGG

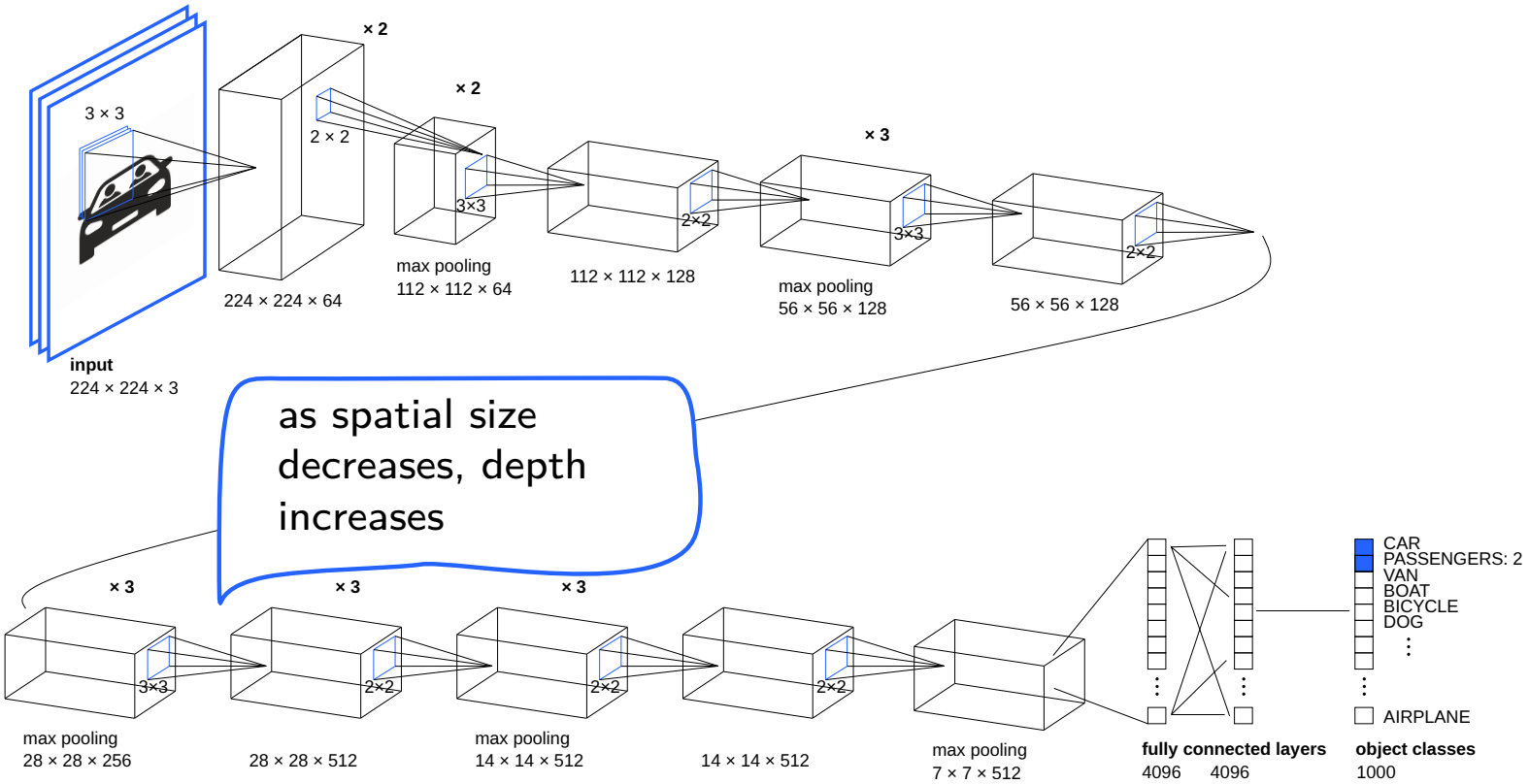
The VGG architecture was introduced by Simonyan and Zisserman (Oxford) in 2014.
The main idea behind this model is simplicity and depth. VGG-16 and VGG-19 has a total of 16 or 19 layers respectively and 138M parameters.



VGG won first and the second places in the localisation and classification tracks in the ImageNet Challenge 2014.

VGG

The VGG architecture was introduced by Simonyan and Zisserman (Oxford) in 2014.
The main idea behind this model is simplicity and depth. VGG-16 and VGG-19 has a total of 16 or 19 layers respectively and 138M parameters.



VGG won first and the second places in the localisation and classification tracks in the ImageNet Challenge 2014.

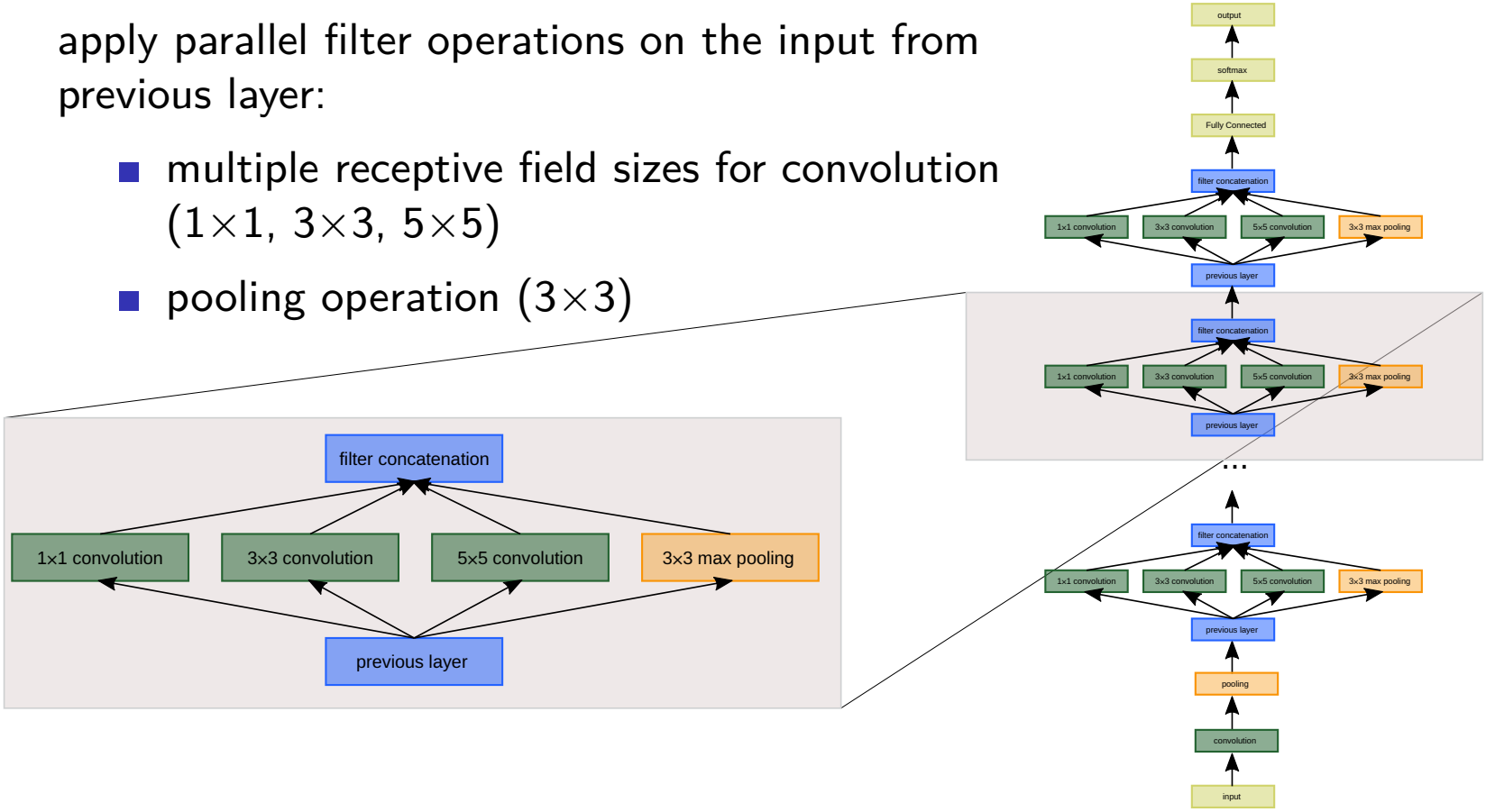
GoogLeNet (Inception-v1)

GoogLeNet, also called Inception-v1 (Szegedy et al. 2014) is built on the concept of a **inception module**: design a good local network topology and then **stack** these modules on top of each other (network within a network)

a naive inception module:

apply parallel filter operations on the input from previous layer:

- multiple receptive field sizes for convolution (1×1, 3×3, 5×5)
- pooling operation (3×3)

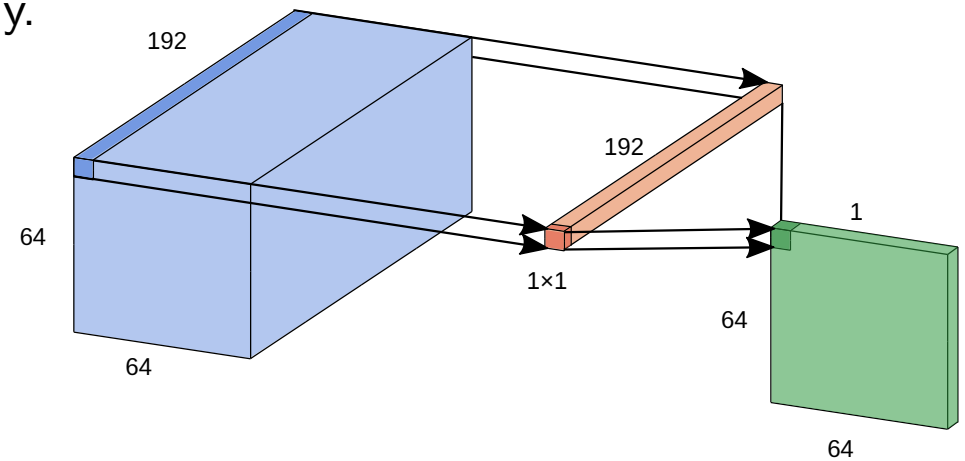


GoogLeNet (Inception-v1)

The **motivation** behind inception networks is to use more than a single type of convolution layer at each layer. It uses for example 1×1, 3×3, 5×5 convolutional layers, and max-pooling layers in parallel; all modules use the same padding.

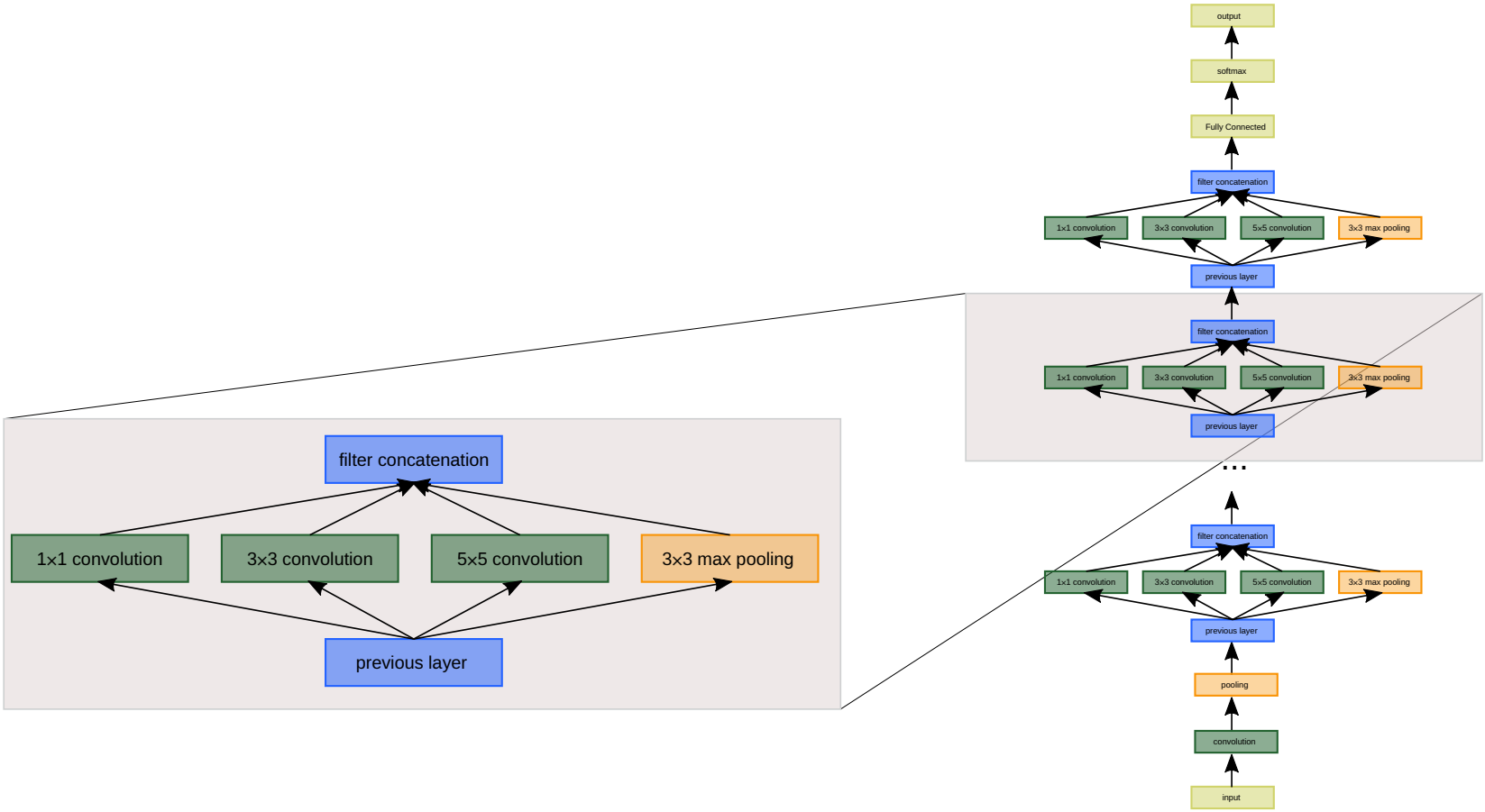
Before taking a more detailed look at GoogleNet, we take a look at 1×1 convolutions:

A 1×1 convolution is used to change the number of channels while keeping the height and the width (spatial information) the same. It mixes channels without introducing more parameters, thus makes deeper networks more efficiently.



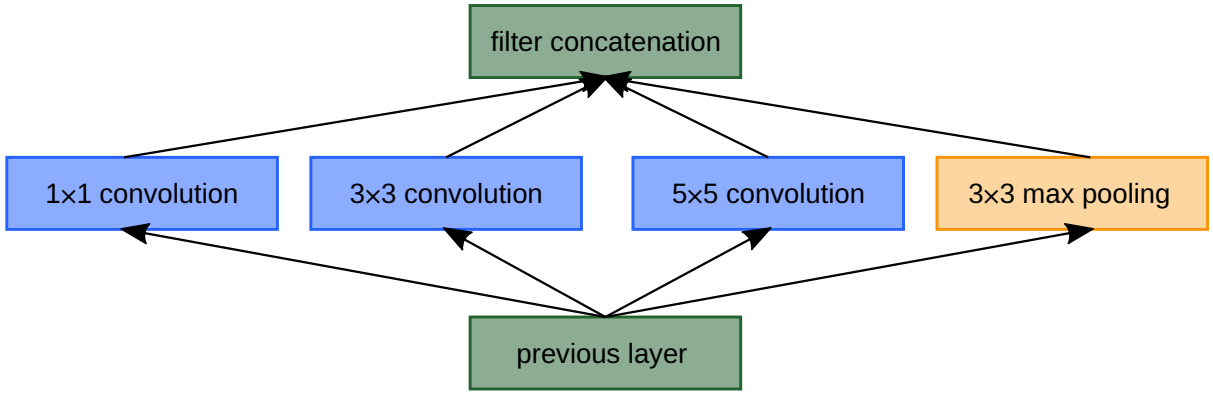
GoogLeNet (Inception-v1)

The inception network is finally formed by concatenating other inception modules. It includes several softmax output units to enforce regularization.



GoogLeNet (Inception-v1)

the problem: **computational complexity**

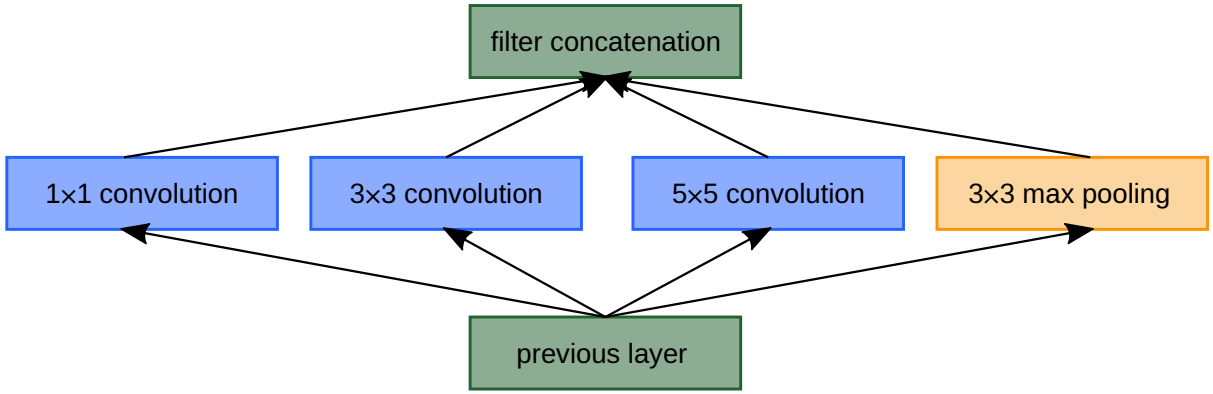


the **convolutional operations**:

e.g. with a module input of $28 \times 28 \times 256$, we get:
1×1 convolution: $28 \times 28 \times 128 \times 1 \times 1 \times 256$ operations
3×3 convolution: $28 \times 28 \times 192 \times 3 \times 3 \times 256$ operations
5×5 convolution: $28 \times 28 \times 96 \times 5 \times 5 \times 256$ operations

GoogLeNet (Inception-v1)

the problem: **computational complexity**

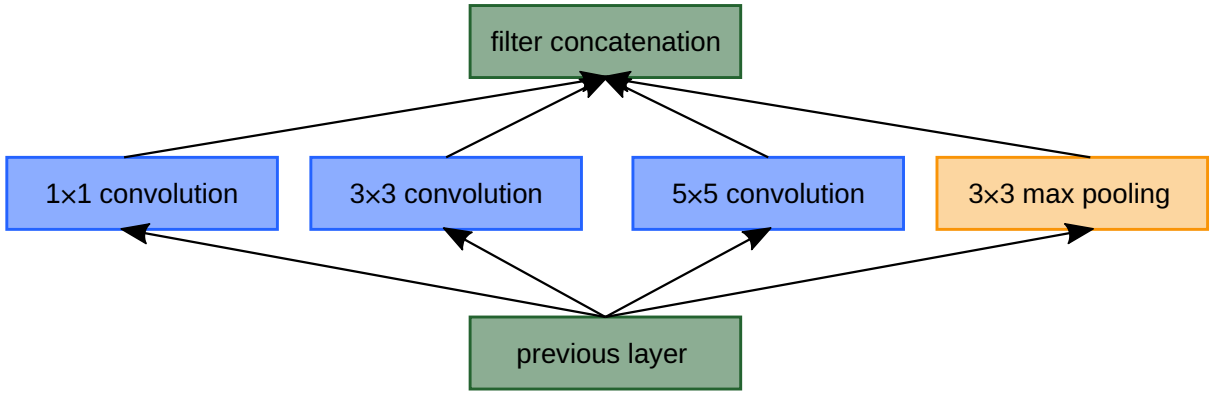


the **convolutional operations**:

e.g. with a module input of $28 \times 28 \times 256$, we get:
1×1 convolution: $28 \times 28 \times 128 \times 1 \times 1 \times 256$ operations
3×3 convolution: $28 \times 28 \times 192 \times 3 \times 3 \times 256$ operations
5×5 convolution: $28 \times 28 \times 96 \times 5 \times 5 \times 256$ operations
$\Rightarrow$ total 854 $\times 10^6$ operations

GoogLeNet (Inception-v1)

the problem: **computational complexity**



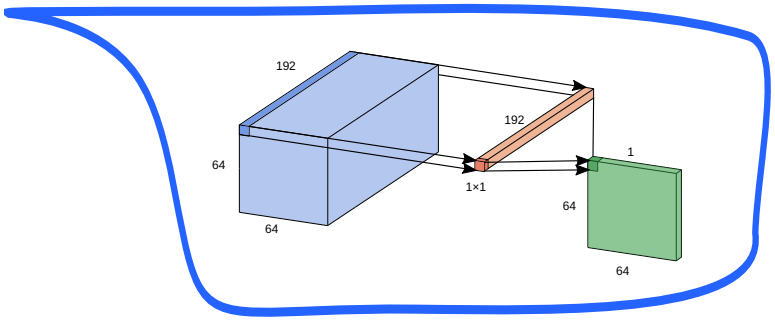
the **convolutional operations**:

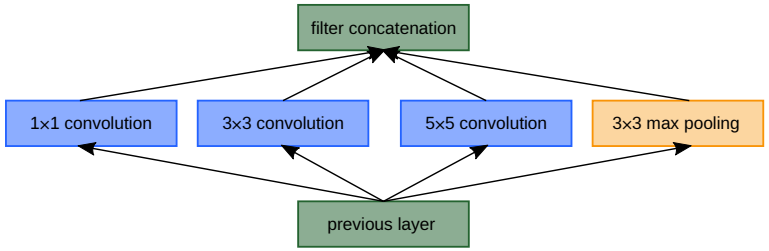
e.g. with a module input of $28 \times 28 \times 256$, we get:
1×1 convolution: $28 \times 28 \times 128 \times 1 \times 1 \times 256$ operations
3×3 convolution: $28 \times 28 \times 192 \times 3 \times 3 \times 256$ operations
5×5 convolution: $28 \times 28 \times 96 \times 5 \times 5 \times 256$ operations
$\Rightarrow$ total 854×10^6 operations

plus: **pooling layer** preserves the feature depth which means total depth after concatenation can only grow at every layer

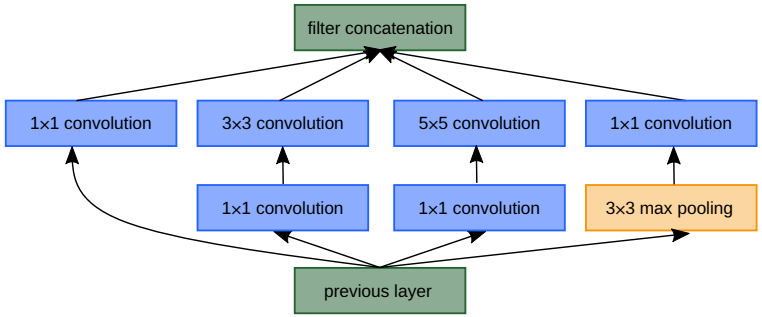
GoogLeNet (Inception-v1)

Solution (Szegedy et al. 2014): **bottleneck layers** that use 1×1 convolutions to reduce feature depth





naive vs.



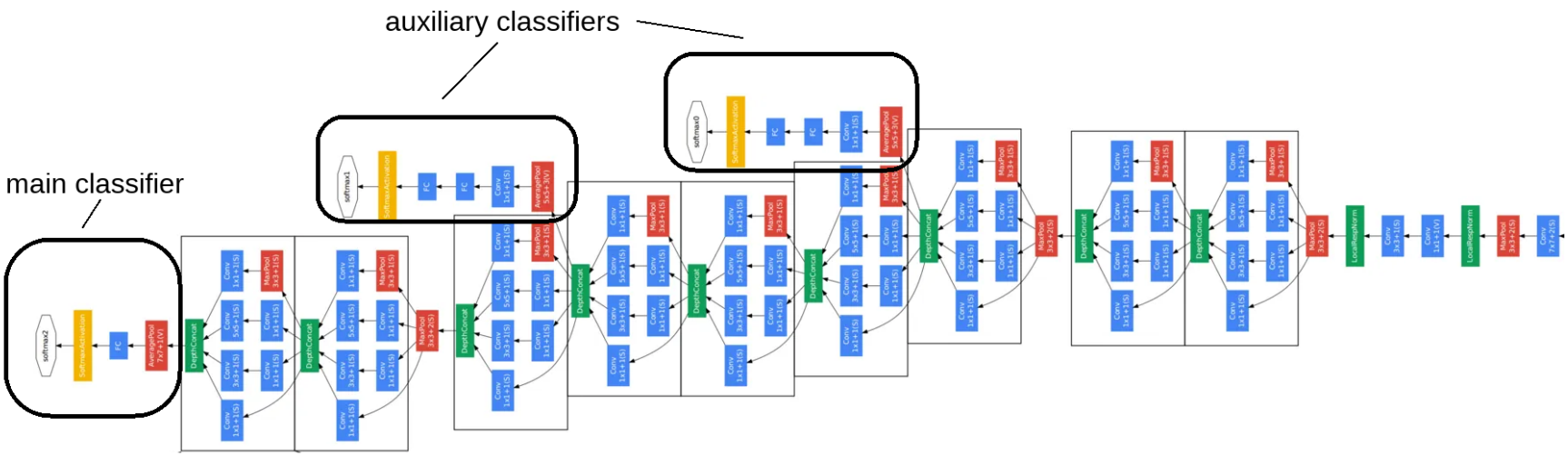
with dimensionality reduction

computationally more efficient:

e.g. with a module input of $28 \times 28 \times 256$, we get: total 358×10^6 operations (compared to 854×10^6 operations for the naive version)

GoogLeNet (Inception-v1)

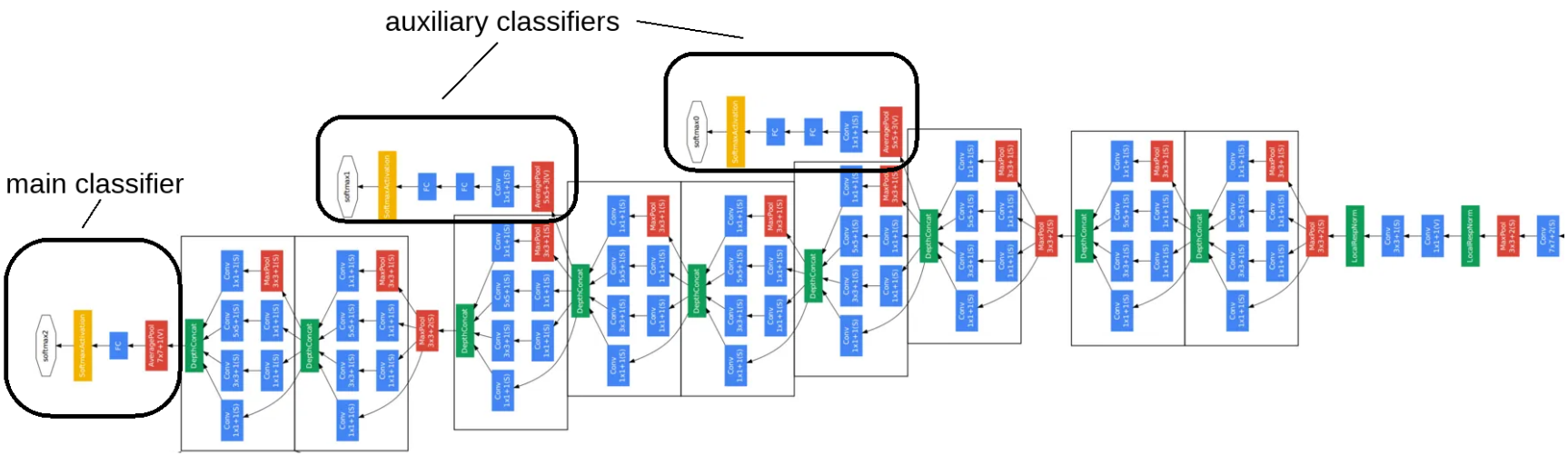
adding more layers comes with a **drawback**: exploding/vanishing gradients



GoogLeNet (Inception-v1)

adding more layers comes with a **drawback**: exploding/vanishing gradients

GoogLeNet solves this by adding **auxiliary classifiers** after the 3rd and 6th inception module to increase the gradient signal that gets propagated back. During training, their loss is added to the network's total loss with a 0.3 weight. At inference time, these auxiliary networks are discarded.

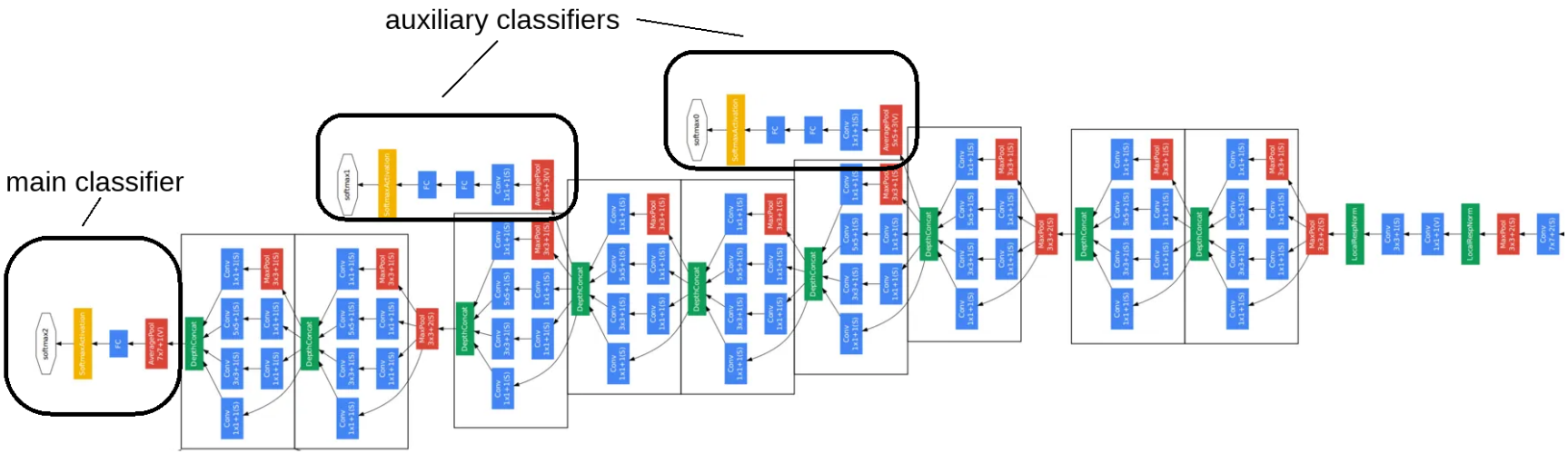


GoogLeNet (Inception-v1)

adding more layers comes with a **drawback**: exploding/vanishing gradients

GoogLeNet solves this by adding **auxiliary classifiers** after the 3rd and 6th inception module to increase the gradient signal that gets propagated back. During training, their loss is added to the network's total loss with a 0.3 weight. At inference time, these auxiliary networks are discarded. The auxiliary classifiers, as well as the last layer of the main classifier, use a softmax activation function (all others use ReLU).

Auxiliary classifiers start with a 5×5 average-pooling, followed by a convolution layer with 1×1 kernel size and two fully connected layers.

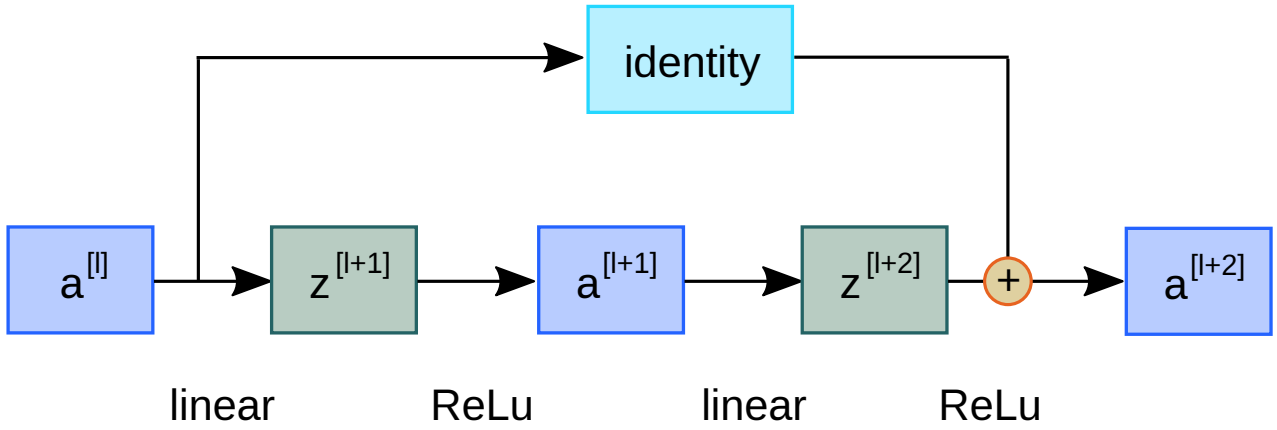


ResNet

Vanishing gradients are a common problem in Deep Learning, leading to weights not updated anymore and therefore, no learning takes place.

ResNet, presented by He et al. (Microsoft), 2015, enables the training of even very deep NNs ($>$ 100 layers): Using network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping.

residual network block:



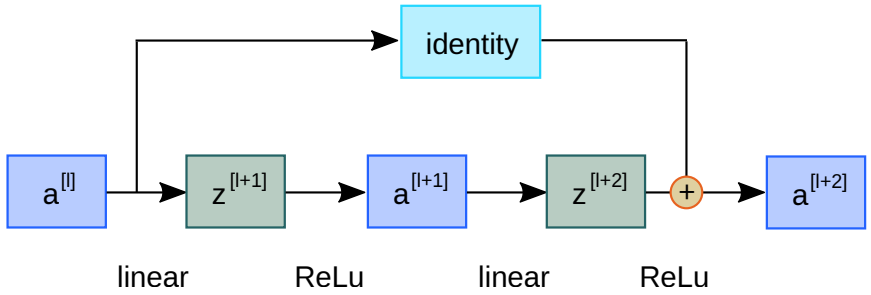
ResNet won ILSVRC 2015 in multiple categories.

ResNet

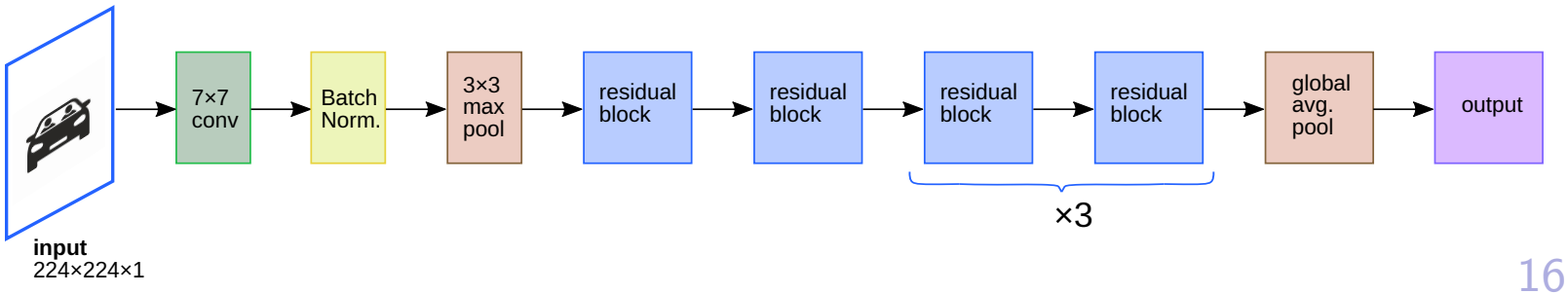
Vanishing gradients are a common problem in Deep Learning, leading to weights not updated anymore and therefore, no learning takes place.

ResNet, presented by He et al. (Microsoft), 2015, enables the training of even very deep NNs (> 100 layers): Using network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping.

residual network block:



Layers are stacked sequentially:



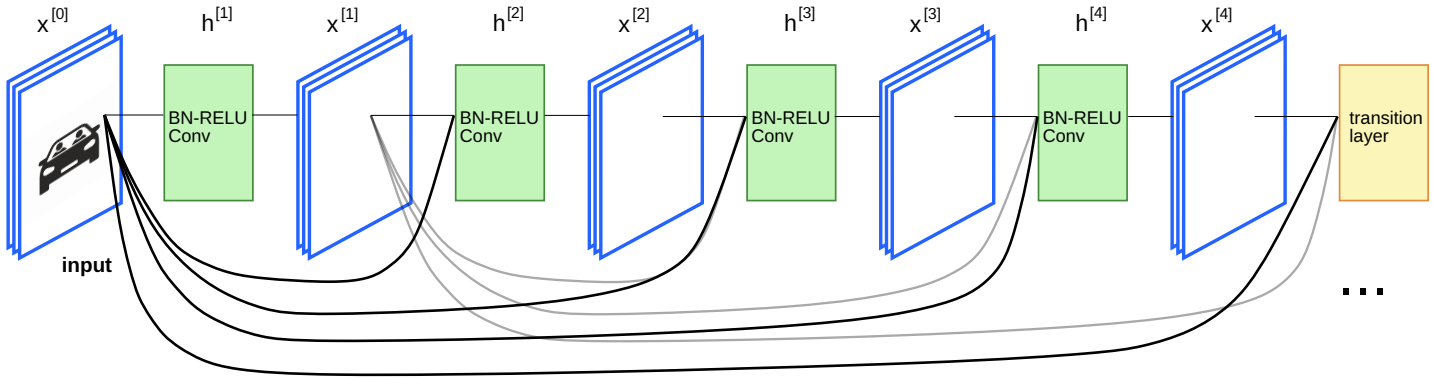
DenseNets

DenseNets (Huang 2016) exploit the potential of the NN through feature reuse. Therefore, there is no need to learn redundant feature maps.

For this reason, DenseNets layers are very narrow (e.g. 12 filters), and they just add a small set of new feature-maps.

Goal: Allow maximum information (and gradient) flow by connecting every layer directly with each other.

Dense Block:

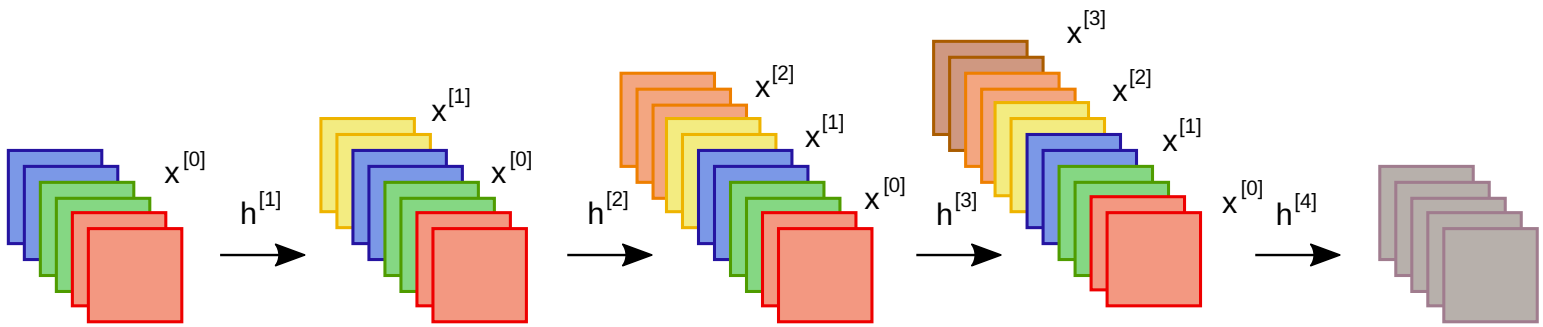


DenseNets

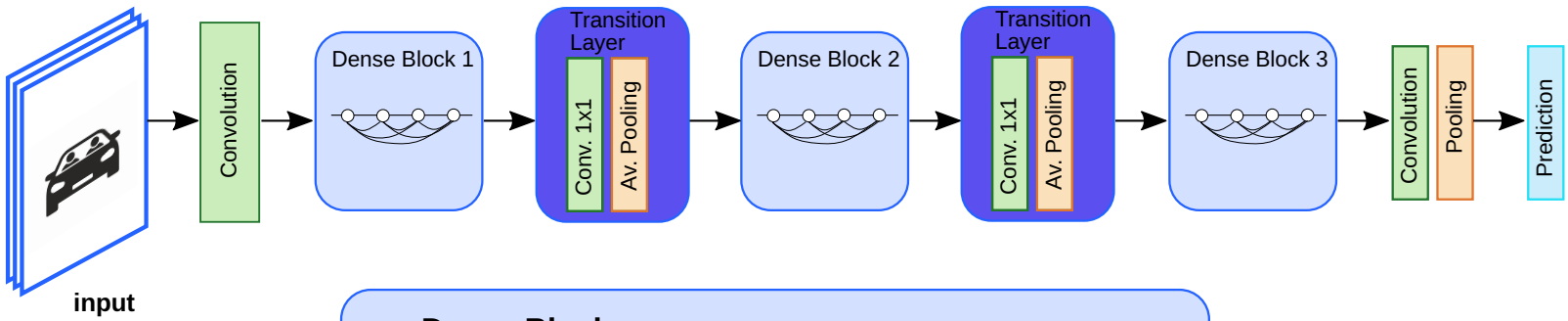
Properties of DenseNets:

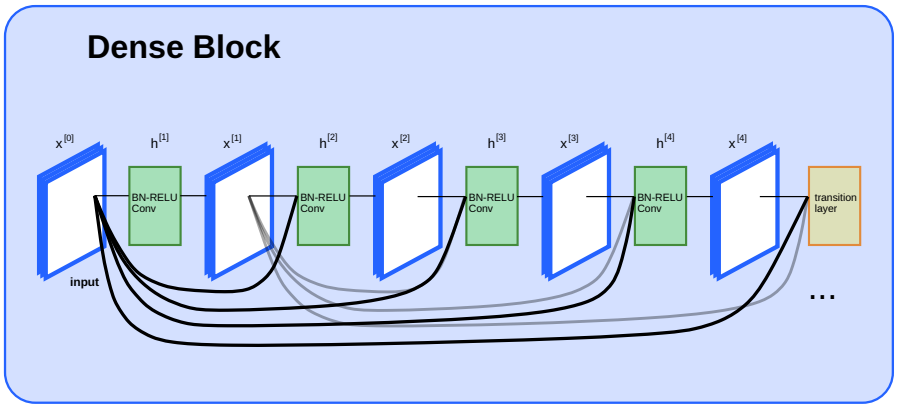
- DenseNets do not sum the output feature maps of the layer with the incoming feature maps, but concatenate them: $a^{[l]} = g([a^{[0]}, a^{[1]}, ..., a^{[l-1]}])$
- dimensions of the feature maps remain constant within a block, but the number of filters changes between them: $k^{[l]} = k^{[0]} + k(l-1)$

Concatenation during forward propagation:



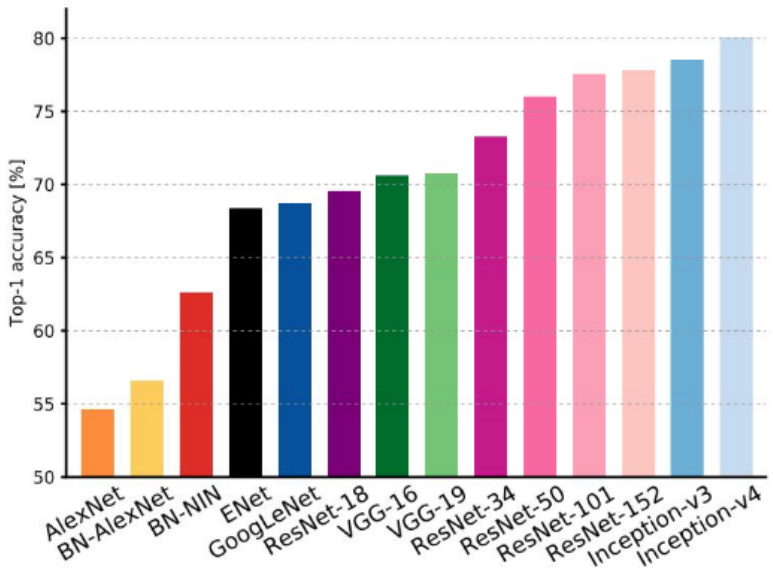
DenseNets

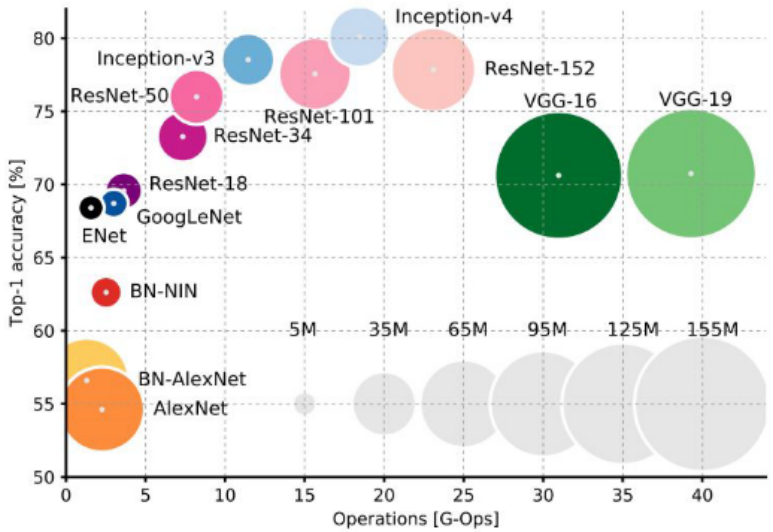




CNN Architectures - Case Studies

comparing complexity





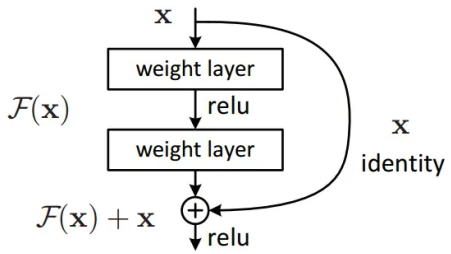
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

CNN Architectures - Summary

CNN Architecture	Default Input	Default Output	Number of Layers	Number of Parameters	Activation Function	specific properties
LeNet-5	$32 \times 32 \times 1$	10	5	60k	tanh	convolution layer
AlexNet	$244 \times 244 \times 3$	1000	8	60M	ReLU	local response normalization
VGG-16	$244 \times 244 \times 3$	1000	16	138M	ReLU	very deep but single-thread
GoogLeNet	$244 \times 244 \times 3$	1000	22	7M	ReLU	inception module, auxiliary classifiers
ResNet-50	$244 \times 244 \times 3$	1000	50	26M	ReLU	batch normalization, residual blocks

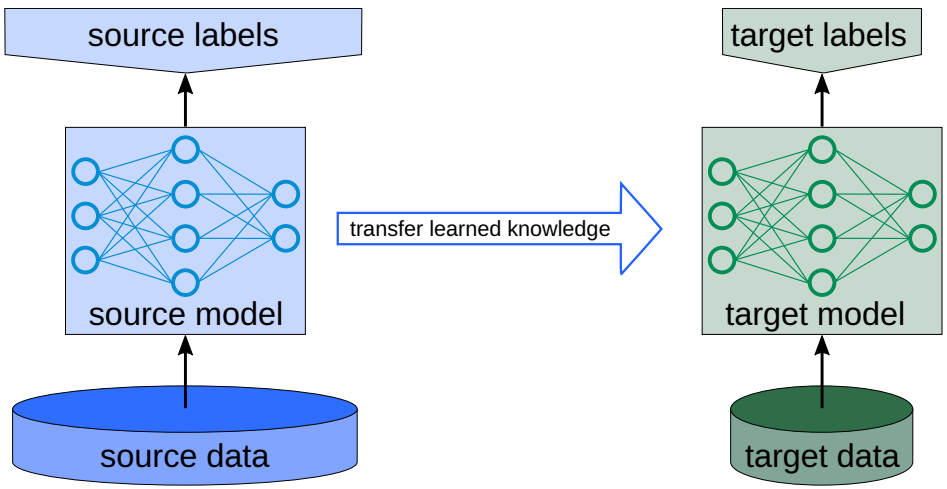
local response normalization:
This implements the idea of lateral inhibition where an excited neuron inhibits its neighbours, leading to contrast in that area.

residual blocks:
Traditionally, each layer feeds into the next layer. In a network with residual blocks, each layer feeds into the next layer and into the layers about 2-3 hops away.



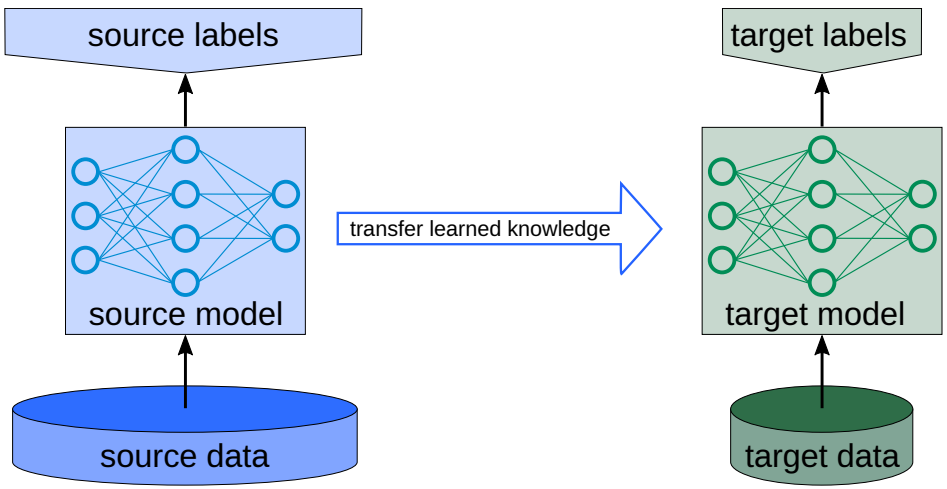
Transfer Learning for Images

The **basic concept** of transfer learning is training on a large dataset and transfer the knowledge to a smaller dataset. It is based on the idea that convolutional layers extract general features applicable across images.



Transfer Learning for Images

The **basic concept** of transfer learning is training on a large dataset and transfer the knowledge to a smaller dataset. It is based on the idea that convolutional layers extract general features applicable across images.



Following is the general outline for transfer learning for object recognition:

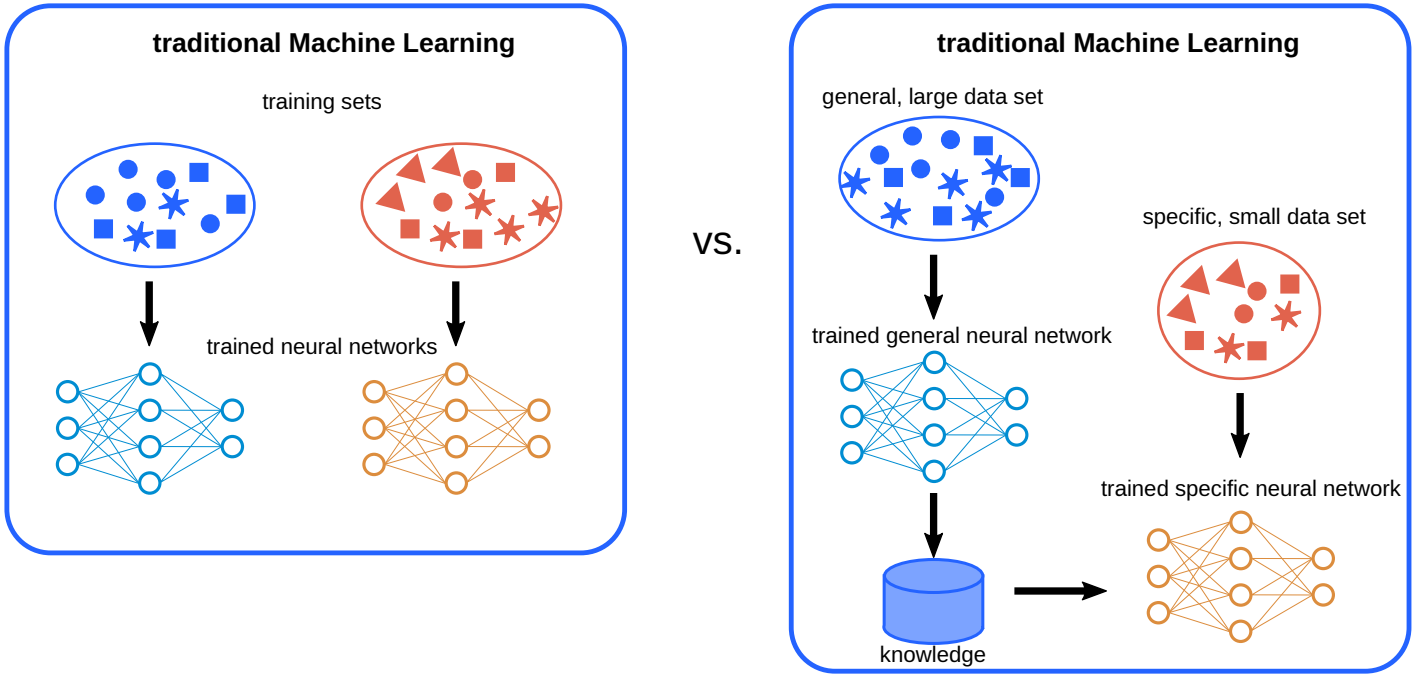
1. Load in a pre-trained CNN model trained on a large dataset.
2. Freeze parameters (weights) in model's lower convolutional layers.
3. Add custom classifier with several layers of trainable parameters.
4. Train classifier layers on training data available for task.
5. Fine-tune hyperparameters and unfreeze more layers as needed.

Transfer Learning for Images

Use pre-trained models, i.e., models with known weights.

Main Idea: Earlier layers of a NN learn low level features, which can be adapted to new domains by changing weights at later layers.

Example: Use a sophisticated huge network (e.g. VGG, Inception etc.) trained on ImageNet. Then retrain it on a few very specific images which might not be present in ImageNet (e.g., rare animals).



Transfer Learning for Images

When does Transfer Learning not work?

In transfer learning we add a classification layer in our new model and train that layer only after freezing feature extraction layers. This will extract high level features for us to differentiate the classes which are learned by the bottom layers.

Transfer Learning for Images

When does Transfer Learning not work?

In transfer learning we add a classification layer in our new model and train that layer only after freezing feature extraction layers. This will extract high level features for us to differentiate the classes which are learned by the bottom layers.

But what if we want low-level features?

Transfer Learning for Images

When does Transfer Learning not work?

In transfer learning we add a classification layer in our new model and train that layer only after freezing feature extraction layers. This will extract high level features for us to differentiate the classes which are learned by the bottom layers.

But what if we want low-level features?

For example, a pre-trained model is able to classify a door but we want to build such a model which can classify whether the door is closed or semi-opened or opened. In this problem statement training only the classification layer of the pre-trained model will not be able to differentiate the classes of our problem and will not give us the required results. We need to retrain more layers of the model or use features from earlier layers which means we are **fine-tuning** our neural network model.

Differential Fine Tuning

The **general idea** to differential fine tune is to train different layers at different rates.

Differential Fine Tuning

The **general idea** to differential fine tune is to train different layers at different rates.

In the pretrained model, the layers closer to the input are more likely to have learned more general features. Thus, we don't want to change them much. Therefore, each "earlier" layer or layer group can be trained at $3\times$ - $10\times$ smaller learning rate than the next "later" one.

Differential Fine Tuning

The **general idea** to differential fine tune is to train different layers at different rates.

In the pretrained model, the layers closer to the input are more likely to have learned more general features. Thus, we don't want to change them much. Therefore, each "earlier" layer or layer group can be trained at $3\times$ - $10\times$ smaller learning rate than the next "later" one.

Moreover, a low learning rate can take a lot of time to train on the "later" layers as they are learning more complex features.

One could even train the entire network again this way until we overfit and then step back some epochs.

Differential Fine Tuning

A recipe for differential fine tuning:

1. **Freeze** the convolutional base.
2. **Train** the fully connected head of the neural network, keeping the convolutional base fixed.
3. **Unfreeze** some later layers in the base network and now train the base network and fully connected network together.

As we're now in the better (smoother) part of the loss surface already, gradients won't be terribly high, but one still needs to be careful about the learning rate. Often a very low learning rate is used.

Transfer Learning vs. Differential Fine Tuning

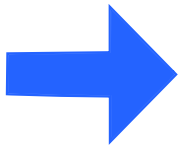
Transfer learning is freezing layers that were previously trained (model) and (optional) adding some new trainable layers. Fine-tuning is unfreezing the entire model you obtained above (or part of it), and re-training it on the new data with a very low learning rate.

When to use Fine-Tuning and when to use Transfer Learning?

Transfer Learning vs. Differential Fine Tuning

When to use Fine-Tuning and when to use Transfer Learning?

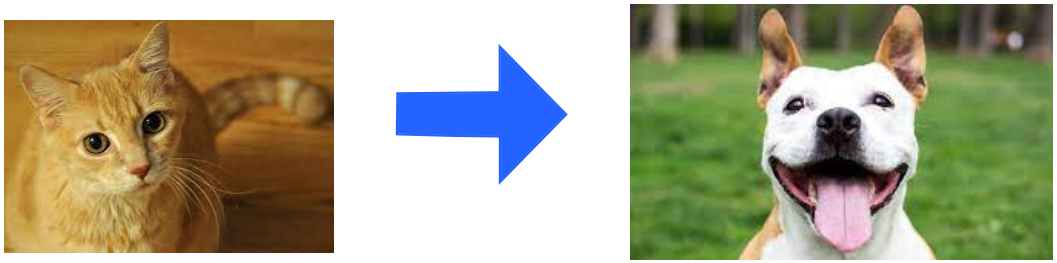
If there are similarities between the source and target model, there is no need to finetune the layers of the pre-trained model. We only need to append a new layer at the end of the network and train our model for the new categories. This is called **deep-layer feature extraction**.



Transfer Learning vs. Differential Fine Tuning

When to use Fine-Tuning and when to use Transfer Learning?

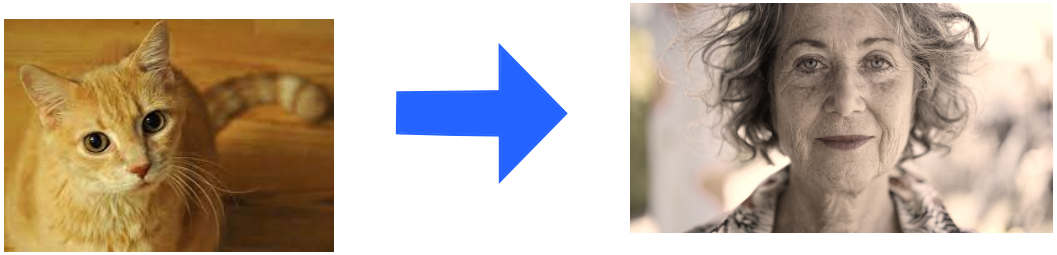
When there are considerable differences between the source and target model, or training examples are abundant, we unfreeze several pretrained model layers except the starting layers which determine edges, corners, etc. Then add the new classification layer and finetune the unfrozen layers with the new examples. This is called **mid-layer feature extraction**.



Transfer Learning vs. Differential Fine Tuning

When to use Fine-Tuning and when to use Transfer Learning?

When there are significant differences between the source and target model, we unfreeze and retrain the entire neural network called **full model fine-tuning**, this type of transfer learning also requires a lot of training examples.



Summary

Deep Neural Network models come with drawbacks such as vanishing or exploding gradients and high computational costs.

State-of-the-Art models help with overcoming these drawbacks.

Transfer Learning helps with training times for data sets with similarities.