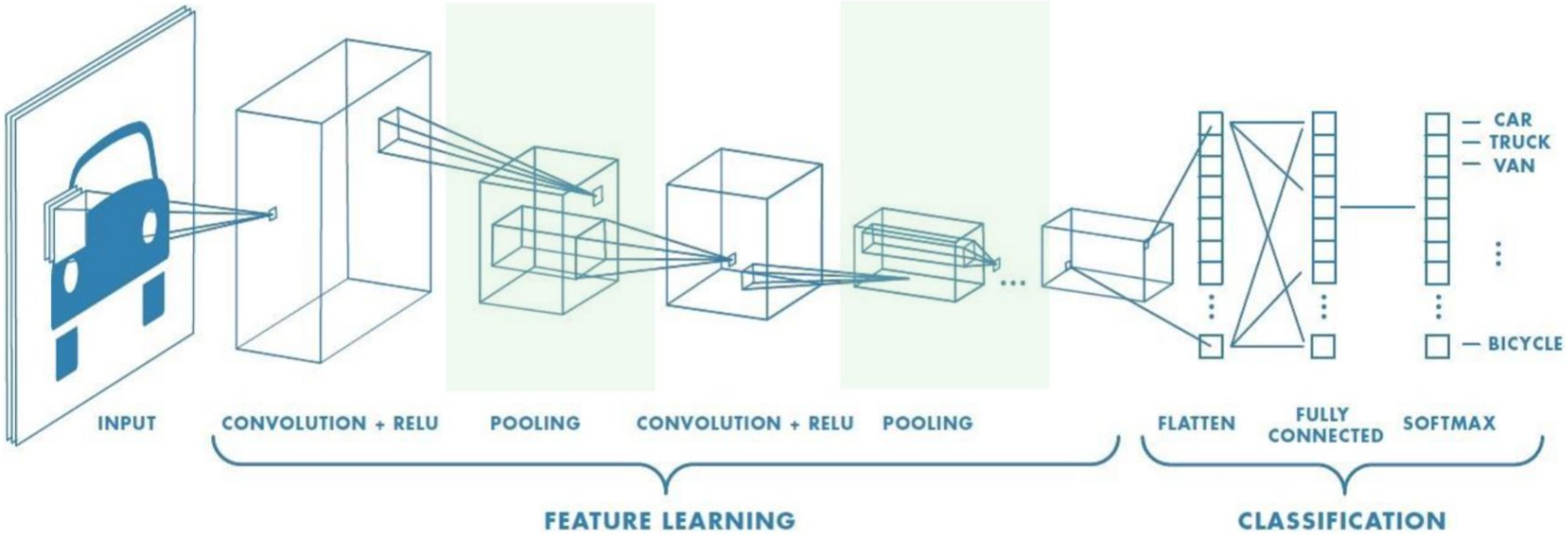
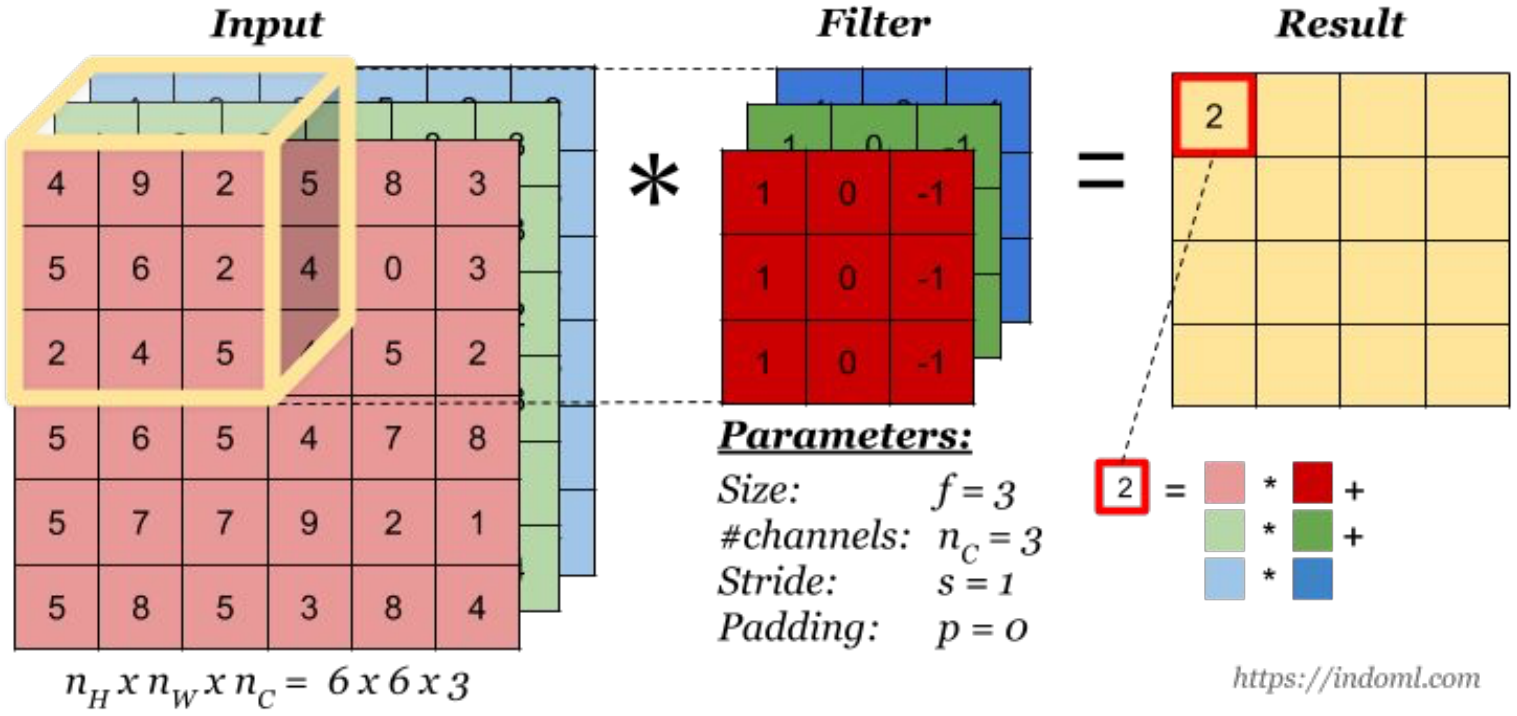
APS360: Applied Fundamentals of Deep Learning

Week 5: Convolutional Neural Network - Part II

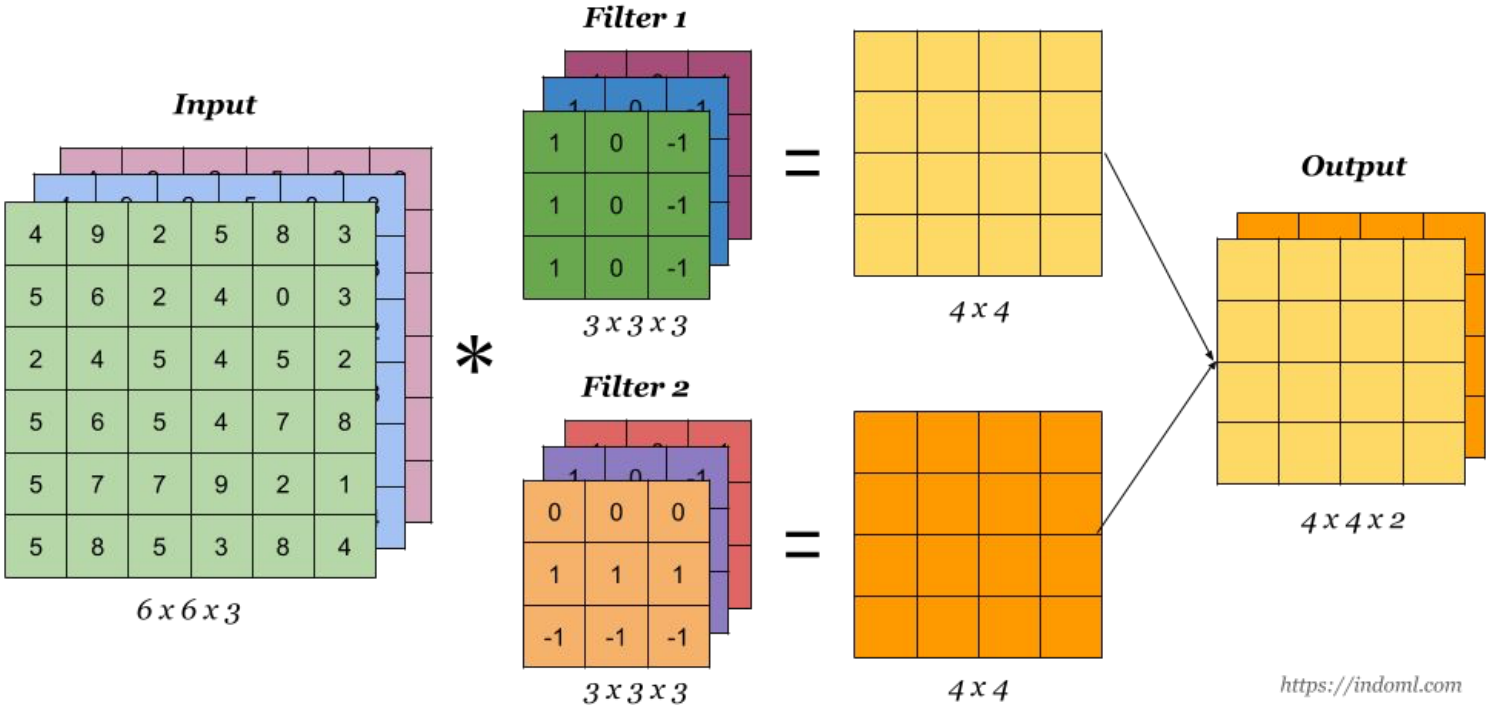


Encoder

Classifier / Head



Note: One value output for convolution, even with multiple channels!


Input
4 9 2 5 8 3
5 6 2 4 0 3
2 4 5 4 5 2
5 6 5 4 7 8
5 7 7 9 2 1
5 8 5 3 8 4
6 x 6 x 3
*
Filter 1
1 0 -1
1 0 -1
1 0 -1
3 x 3 x 3
Filter 2
0 0 0
1 1 1
-1 -1 -1
3 x 3 x 3
=
4 x 4
=
4 x 4
Output
4 x 4 x 2
https://indoml.com

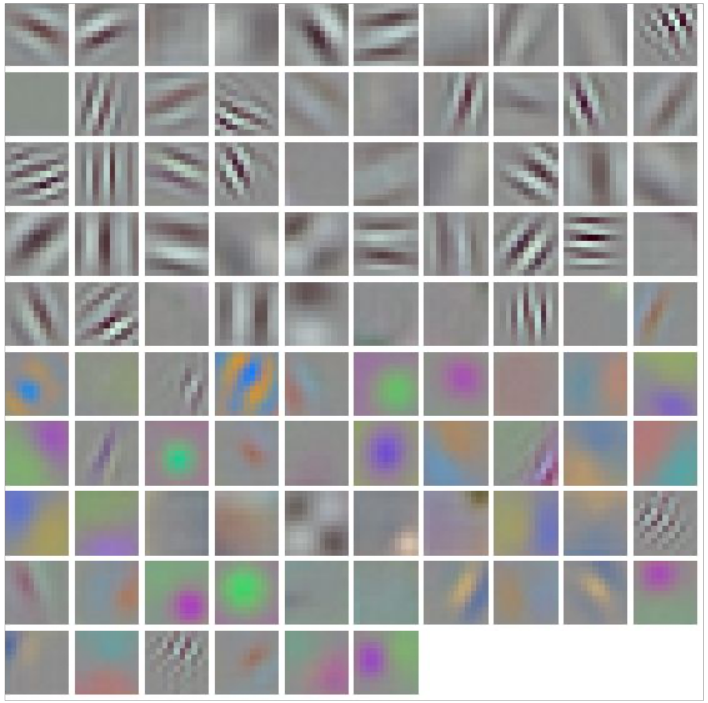
Visualizing Convolutional Filters

What do CNN Filters/Feature maps Look Like?

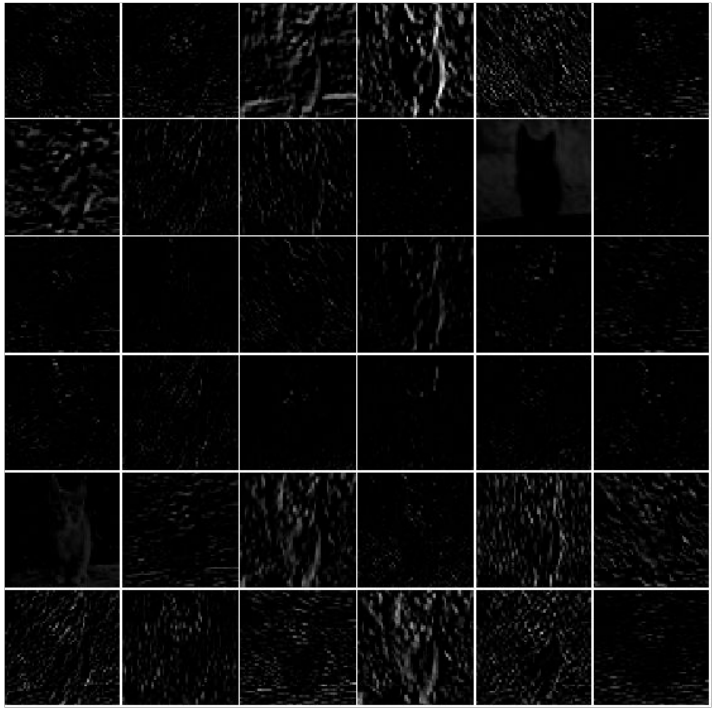
Input image

AlexNet

Conv1 Filters



Conv1 Output

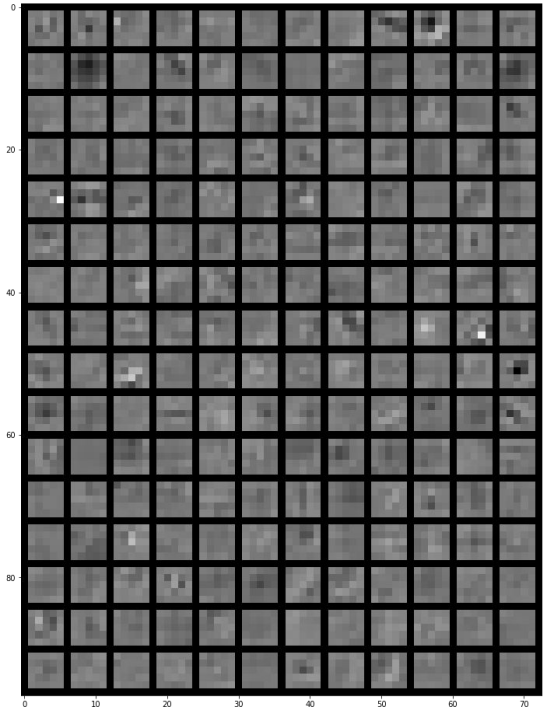


What do CNN Filters/Feature maps Look Like?

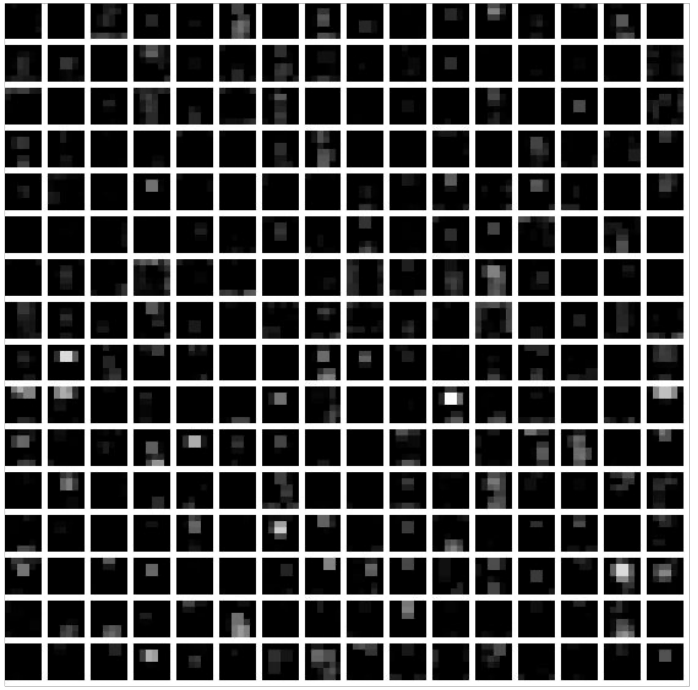
Input image

AlexNet

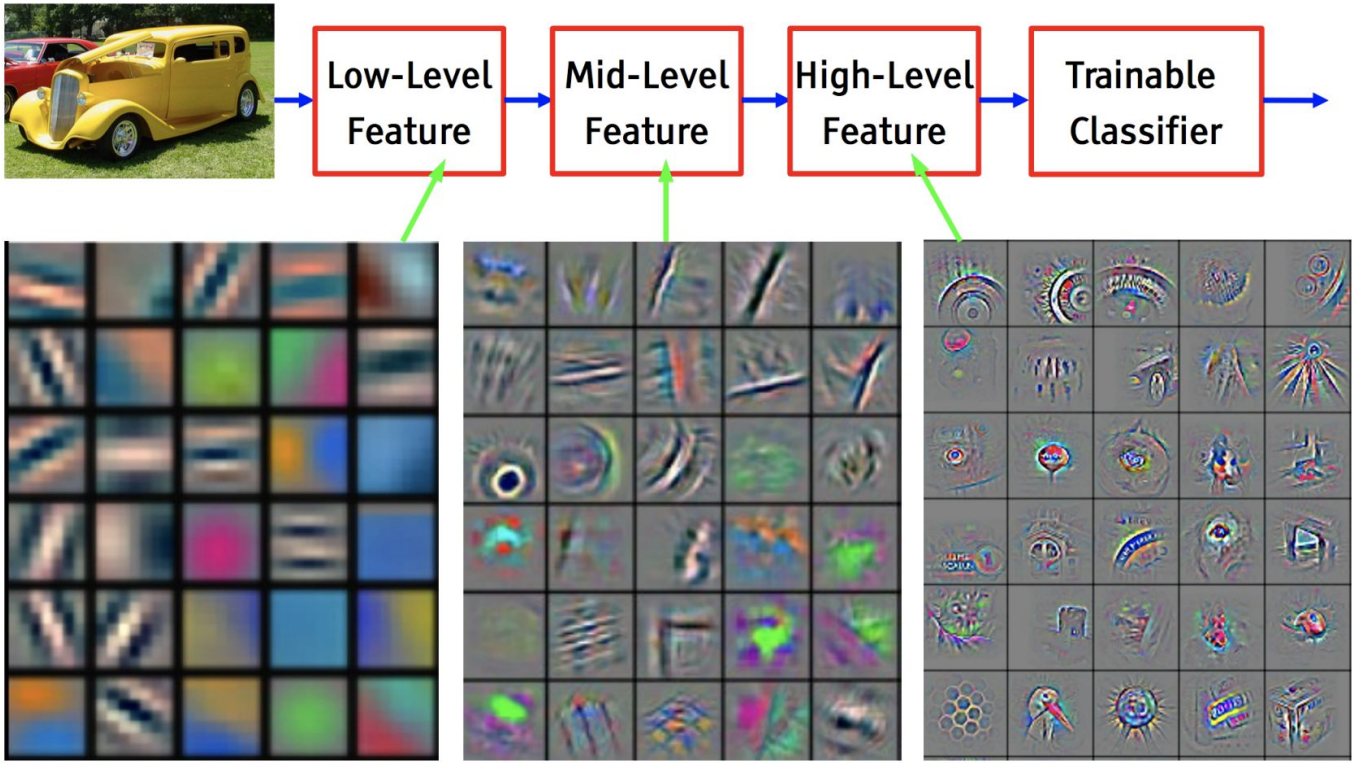
Conv2 Filters



Conv5 Output



What features do CNNs learn?



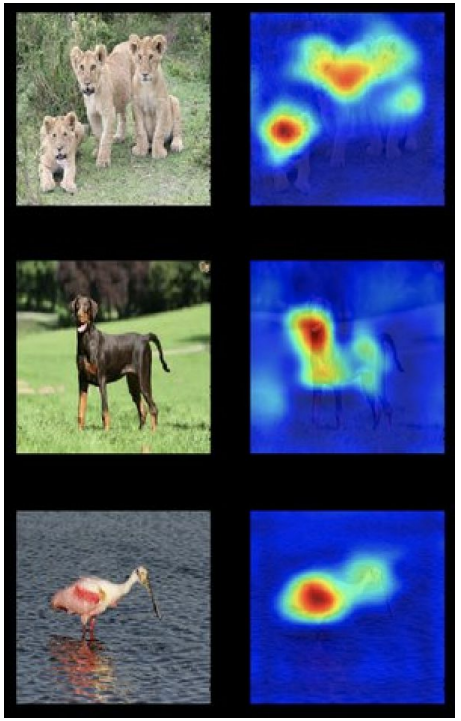
source : Le Cun, 2015

What features do CNNs learn?

Saliency maps : Use gradients of the output over the input to highlight the areas of the images which are relevant for the classification.

1. Feed the image to the network
2. Compute the gradients back to the input image
3. Take the maximum value of absolute gradients across channels
4. Visualize

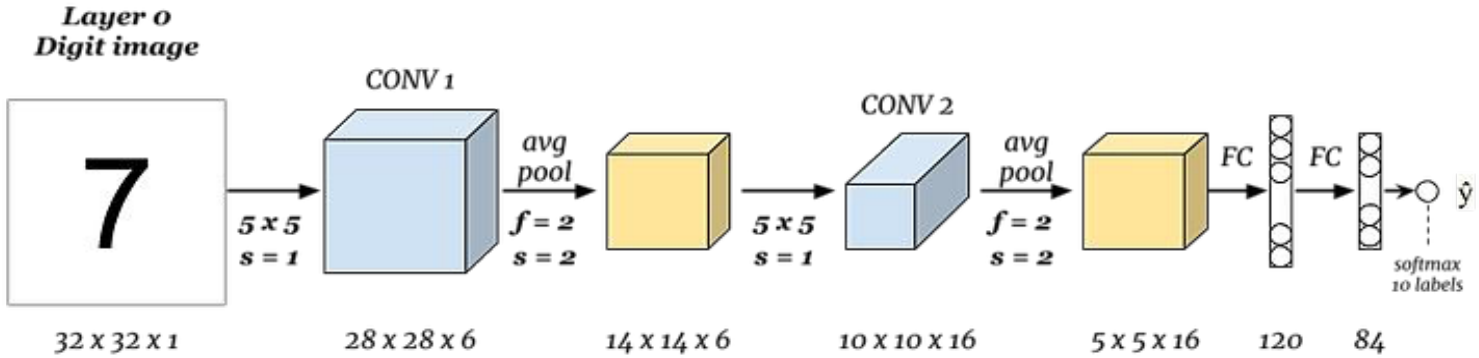
Unfortunately outside giving some intuition, these are not practically very useful, and sometimes even misleading



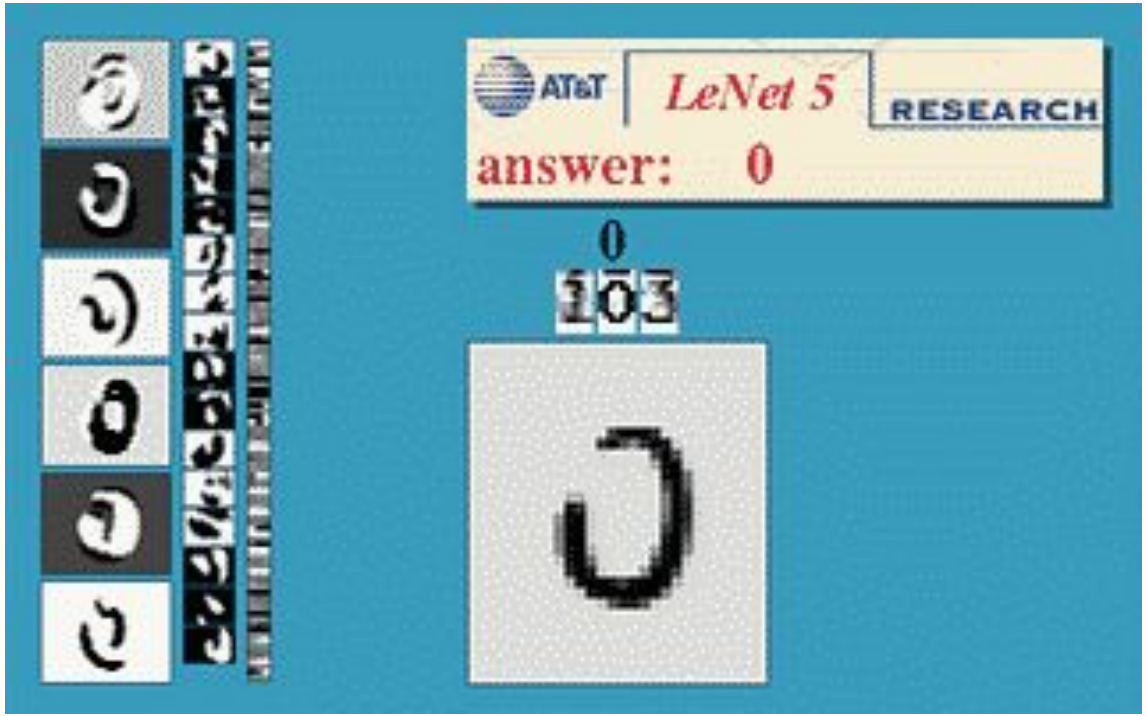
CNNs in Pre-Deep Learning Era

The Original CNN: LeNet

- Convolutional Neural Networks were first introduced by Yann LeCun in 1989
  - Based on earlier "Neocognitron", but little used model (Fukushima, 1980)
- Several variants, mostly we refer to LeNet-5 (above, 1998)
  - 7 layers total, 2 convolutional, 2 subsampling (i.e. pooling), 3 fully-connected

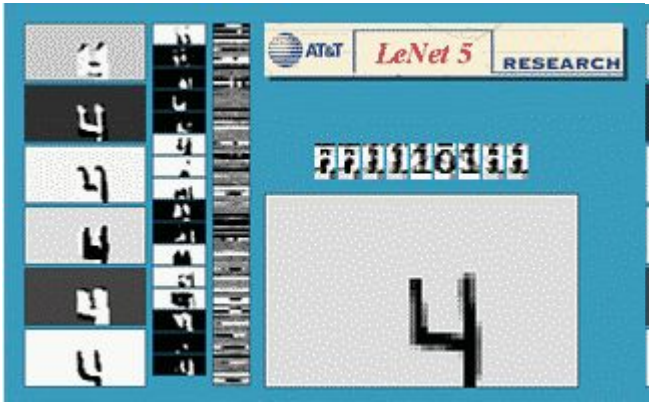


The Original CNN: LeNet Digit Recognition

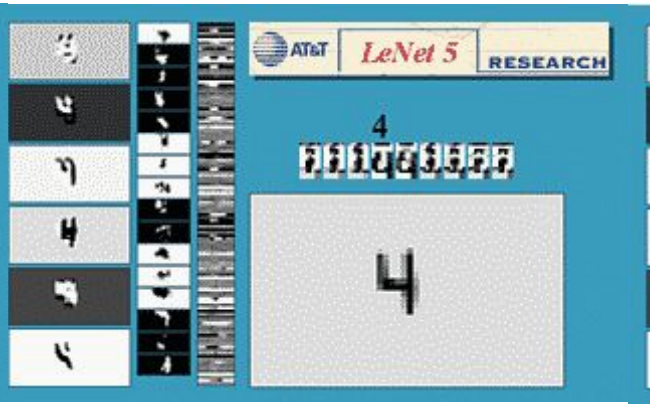


http://yann.lecun.com/exdb/lenet/index.html

Translation Invariance

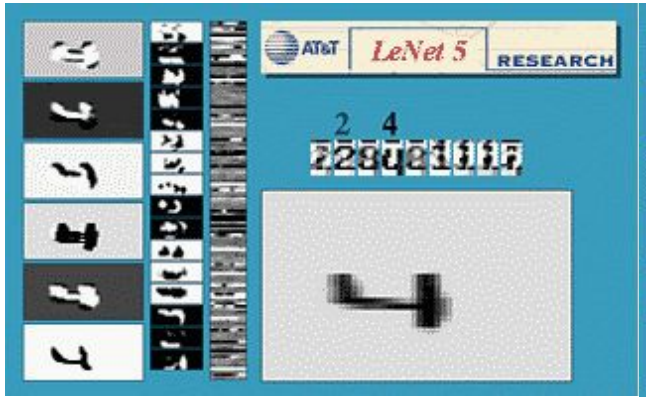


Scale Invariance



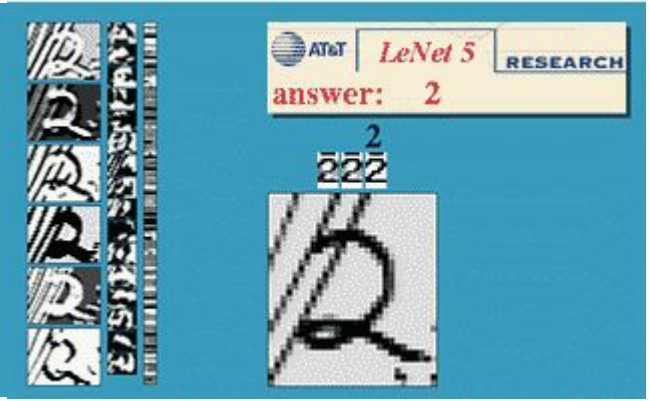
Rotation Invariance

Squeeze/Stretch Invariance



Stroke Width Invariance

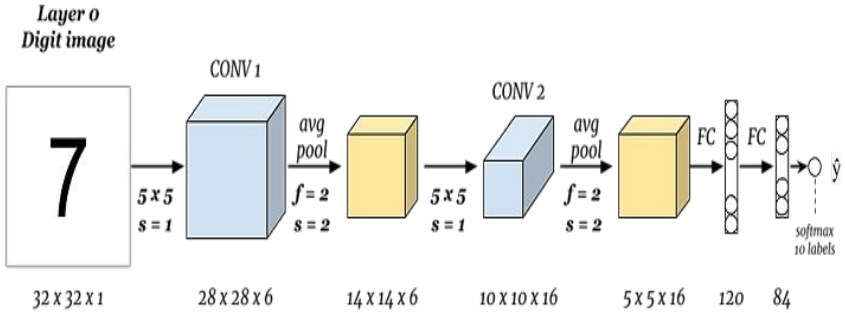
Noise Invariance



LeNet-5: Pytorch Implementation

```
class LeNet5(nn.Module):
    def __init__(self):
        super(LeCun, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(5 * 5 * 16, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 5 * 5 * 16)
        x = F.tanh(self.fc1(x))
        x = F.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```



Modern Architectures

ImageNet Large Scale Visual Recognition Challenge

- **Pascal** VOC was ~20,000 images with 20 classes (2006 - 2009)
- **ImageNet** was the first large-scale image dataset (14 million images)
- **ILSVRC** dataset based on ImageNet
  - 1 Million training images
  - 1000 different classes!
  - 50k validation, **test set (never released)**
- When we say ImageNet we mean ILSVRC

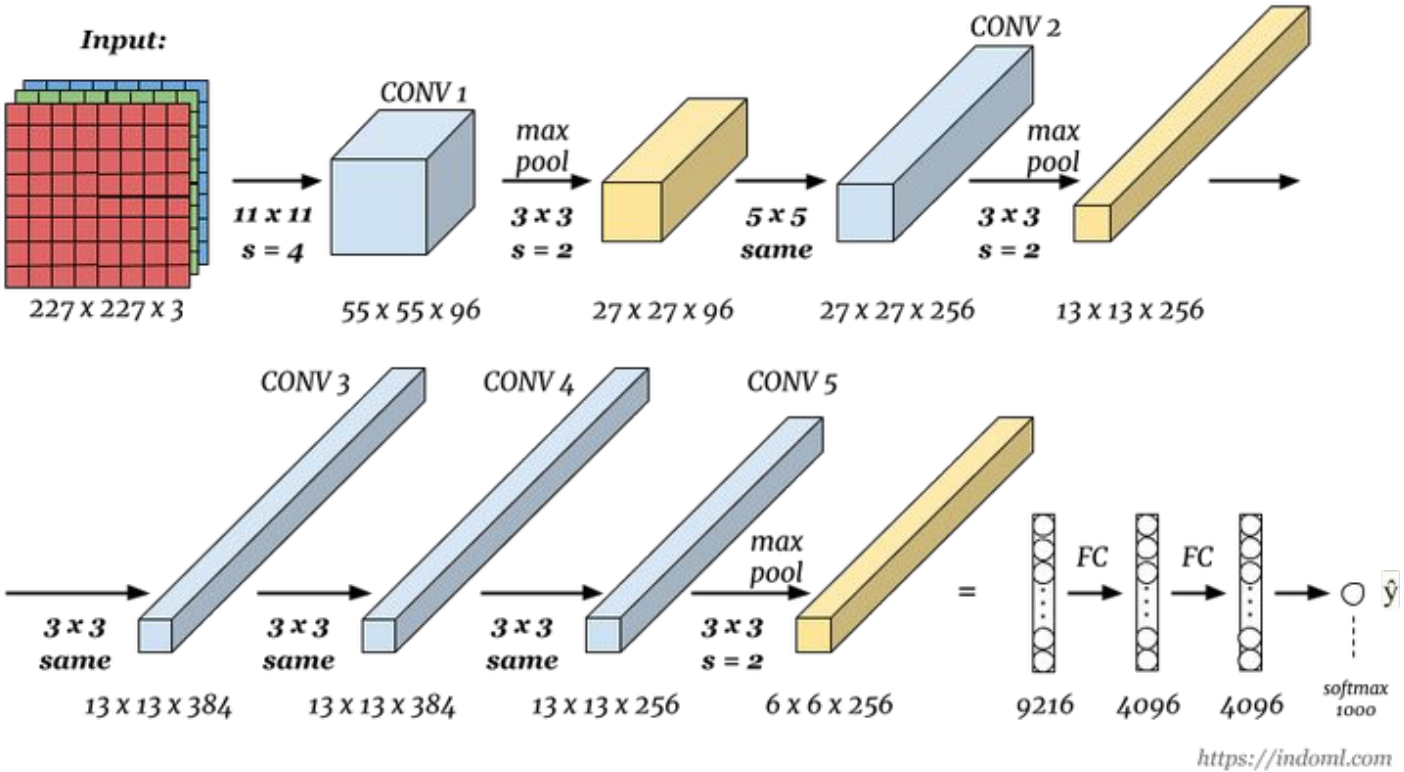
AlexNet

- ILSVRC challenge ran from 2010.
- Like Pascal VOC, every year saw the new winner improve accuracy by ~1-2%
- CNN entry (AlexNet) in 2012 improved accuracy over previous year by ~10%

  This is when "Deep Learning" began

ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS 2012. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

AlexNet



ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS 2012. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

AlexNet: What's Different from LeNet-5?

Deep Learning is differentiated from vanilla **Neural Networks** mostly in the changes between LeNet-5 and AlexNet:

- Much **larger training datasets** (e.g. ImageNet)
- Vast **increase in compute** /GPU acceleration (imagine 1989 PC v.s. 2012!)
- Much **larger model size/more layers** , enabled by both of the above!

AlexNet Training/Architecture Improvements:

- Large number of convolutional layers (i.e. deeper model)
- Use ReLU activation functions instead of sigmoids
- Dropout, **data augmentation**

AlexNet: Training

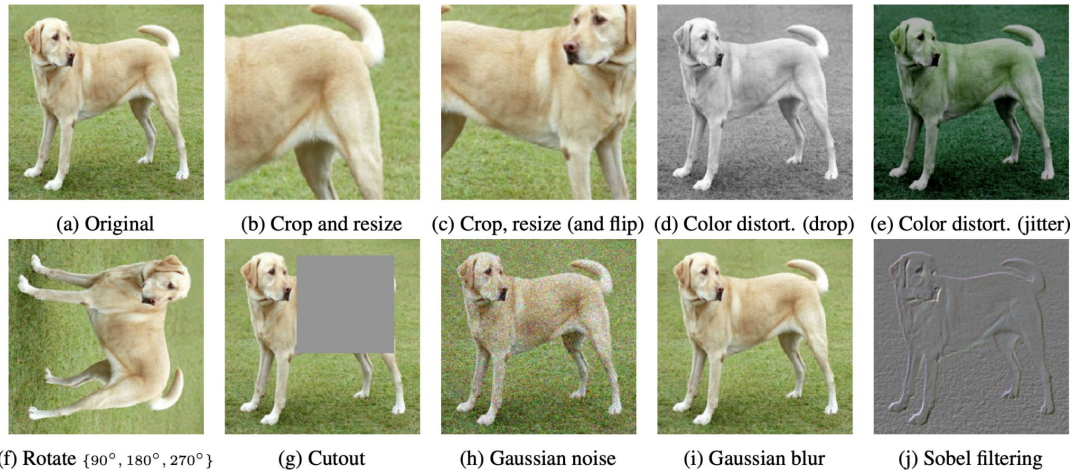
- ~60 Million parameters!
- Used GPUs to accelerate compute: 2 Nvidia GTX 580 GPUs
- 5-6 days to train over 90 epochs
- Optimized with SGD + Momentum
- Uses **weight decay**, **dropout** & **data augmentation** to improve generalization
- Learning rate schedule decreased learning rate 3 times over training

Data Augmentation

Apply class-preserving transformations to the input

- Increases training data
- Helps generalization by learning internal representation of transformations

Used by AlexNet (and all other CNNs)



Generalization and Depth

AlexNet and following models showed that **increased depth improved generalization** on ILSVRC, and other tasks

However, training very deep models often failed!

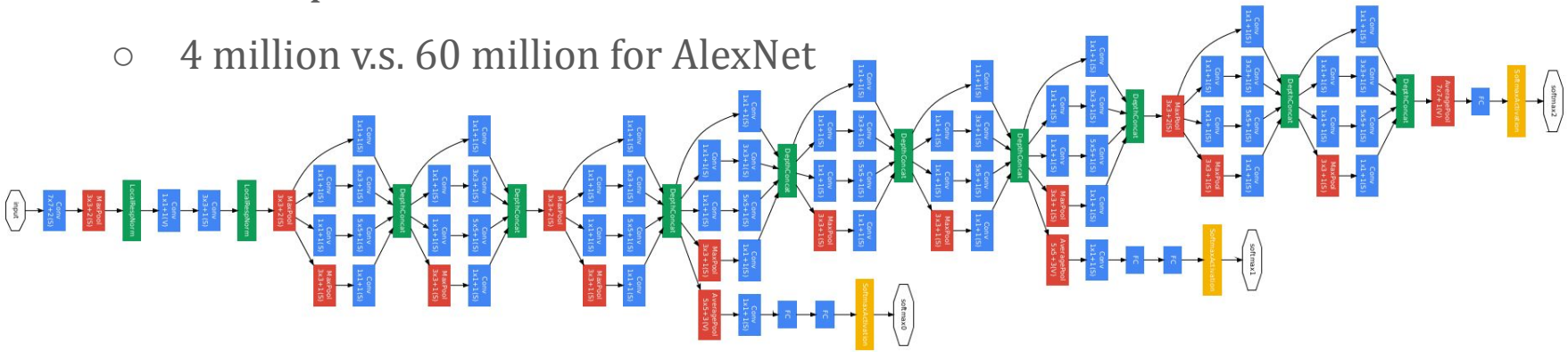
This was due to **vanishing or exploding gradients**

Most important improvements in past 10 years have been to address this:

- Improved initialization for **ReLU**s
- Normalization (e.g. **Batch Normalization**)
- **Residual connections**

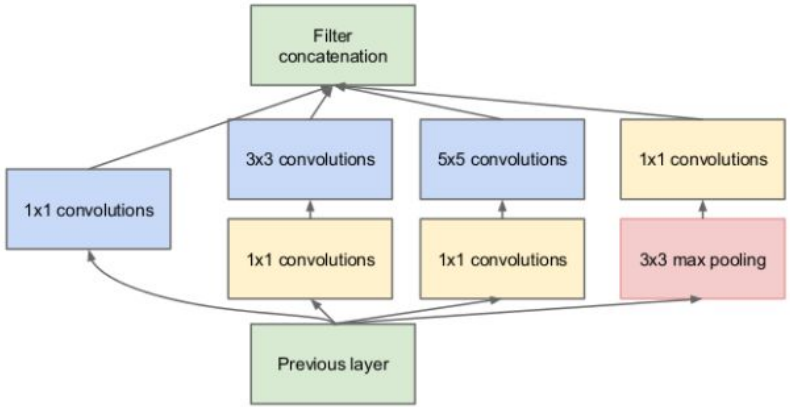
GoogLeNet (Inception)

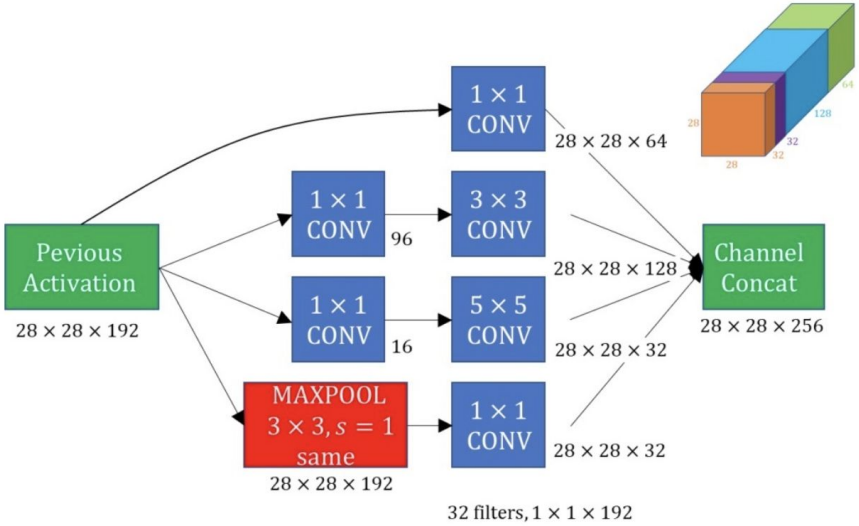
- 2014 ILSVRC winner, 6.67% Top-5 error
  - Human gets ~5.1%
- Primary motivation was to go deeper
  - 22 convolutional layers
- Much more parameter efficient than AlexNet
  - 4 million v.s. 60 million for AlexNet



Inception Block

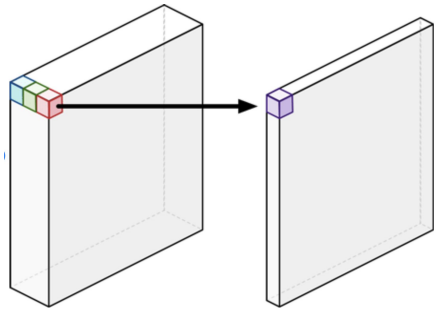
- Uses a mixture of 3x3, 5x5 and 7x7 filters on one layer
- We don't need large 7x7 (or 11x11 as in AlexNet) to learn most important filters.
- We can use mostly 3x3, and add a few larger filters





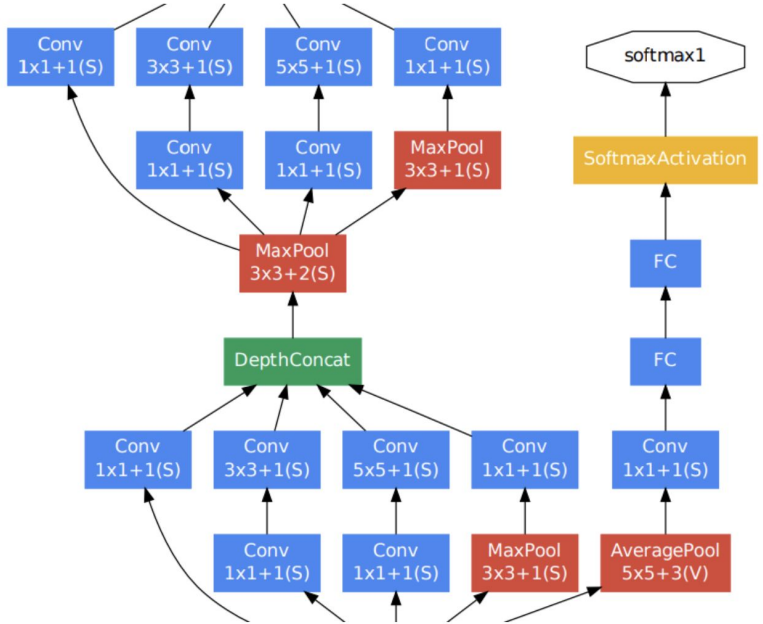
Pointwise (1x1) convolution

- Pixel-wise linear transformations!
  - Originally used in a model called "Network-in-Network"
  - With a non-linearity, they are non-linear pixel-wise transformations
- Learn to map CNN feature maps into a lower or higher dimensional space
  - Good for learning compact representations/compression
- Used in all modern CNN architectures (except VGG)



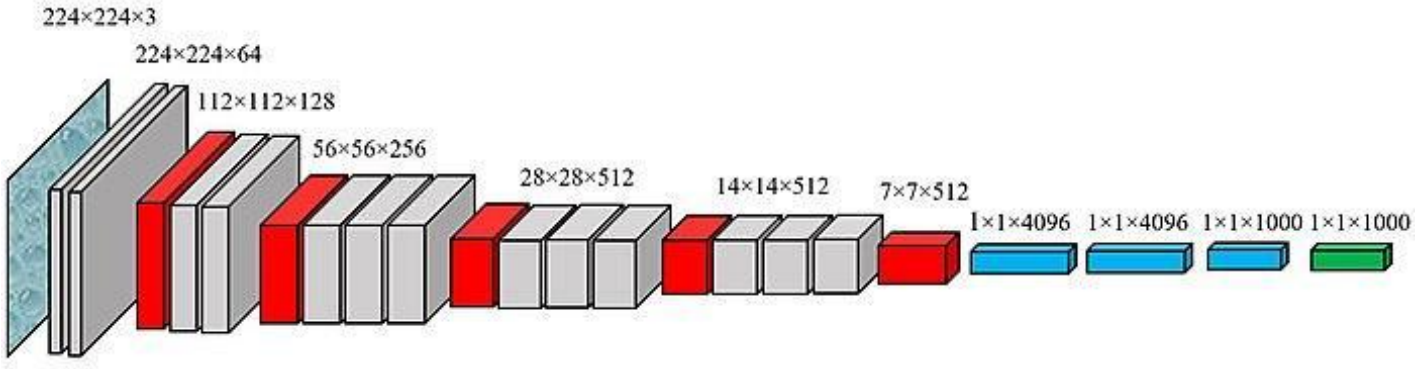
Auxiliary Loss

- Inception network is pretty deep → subject to the vanishing gradient problem.
- Solution → **intermediate classifiers**
- Adding classifiers in the intermediate layers such that the final loss is a combination of the intermediate losses and the final loss.



VGG (Visual Geometry Group, Oxford)

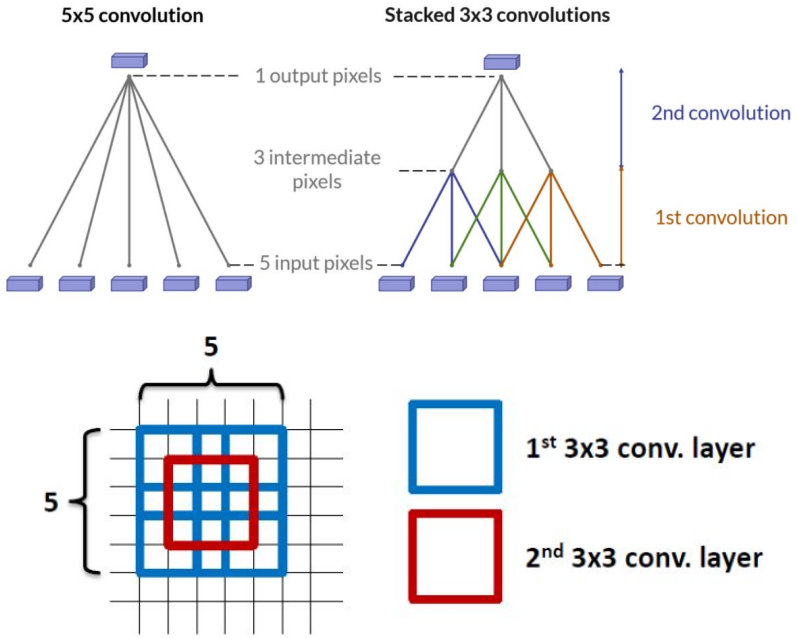
- 2014 ILSVRC classification 2nd place, 7.3% Top-5 error
  - However, won parallel ILSVRC localization challenge
- Proposed Models with 11, 13, 16, and 19 layers
- Very simple architecture, easy to understand/extend
- Very large number of parameters: 138 Million v.s. 60 Million for AlexNet!



VGG (Visual Geometry Group, Oxford)

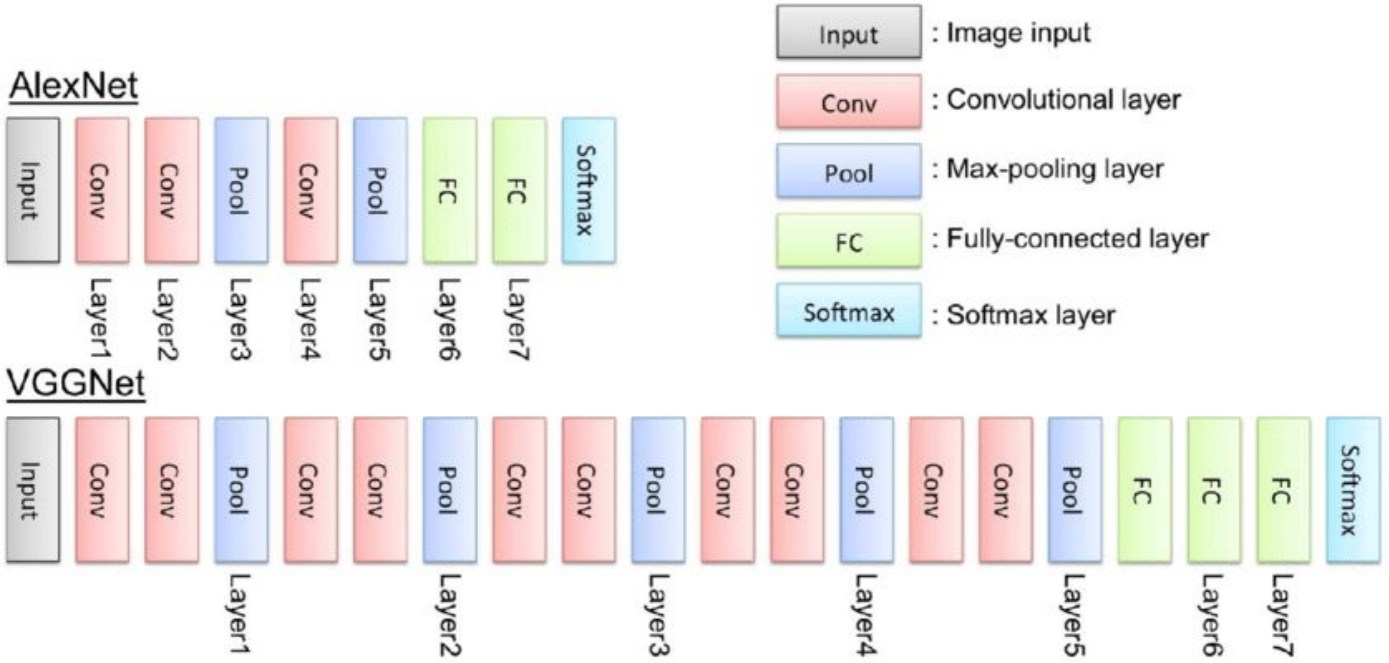
VGG was a very impactful paper:

- Simple architecture made of simple stacked blocks
- We only need 3x3 filters
  - Authors pointed out that stacked 3x3 filters can approximate any larger-sized convolution, more efficiently
  - Since VGG almost all CNNs use mostly/exclusively 3x3 filters!
  - The data augmentation used by VGG is very commonly used



https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d02355543a11

AlexNet v.s. VGG



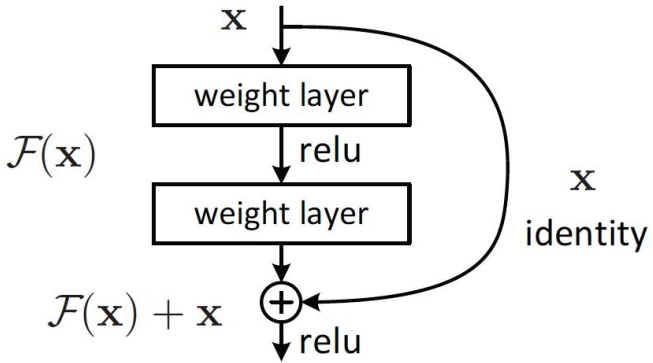
Residual Networks

Even with **Batch Normalization** and **ReLUs** training very deep networks fails

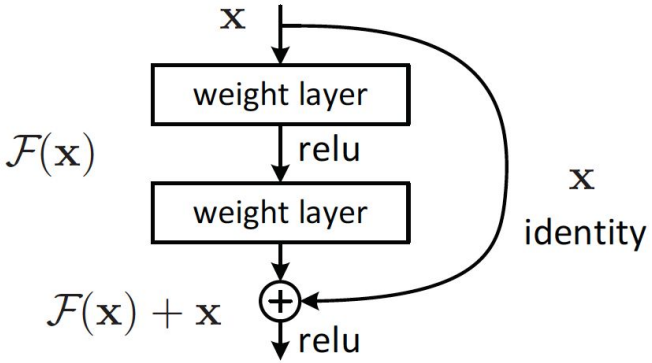
Uses **skip connections** to provide deeper layers more direct access to signals, which otherwise might be lost to vanishing gradients

ResNet won ILSVRC 2015 with 3.57% error

- Model had **152 layers** ,
- Better than Human baseline!



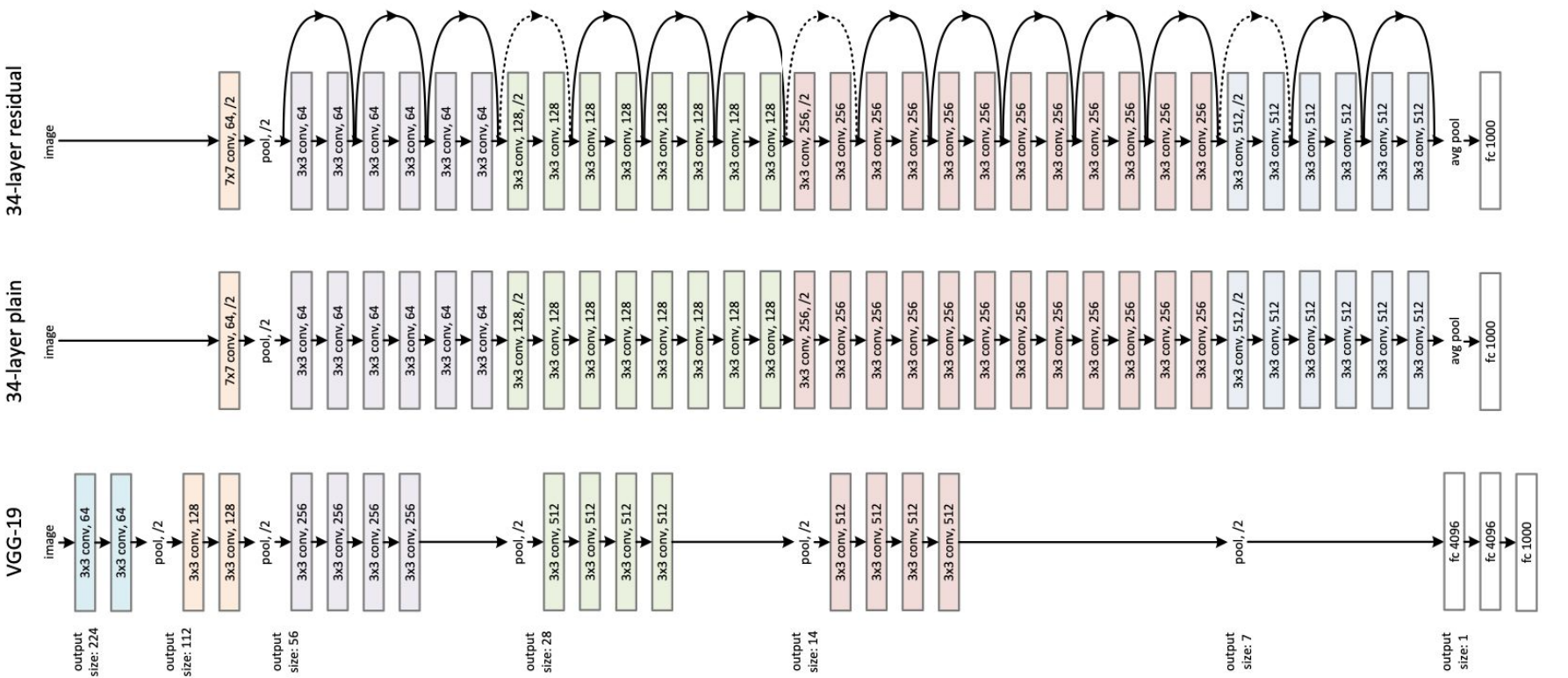
Residual Networks: Skip Connections




```
# normal layer:
next_activation = layer(activation)

# residual layer
next_activation = activation + layer(activation)
```

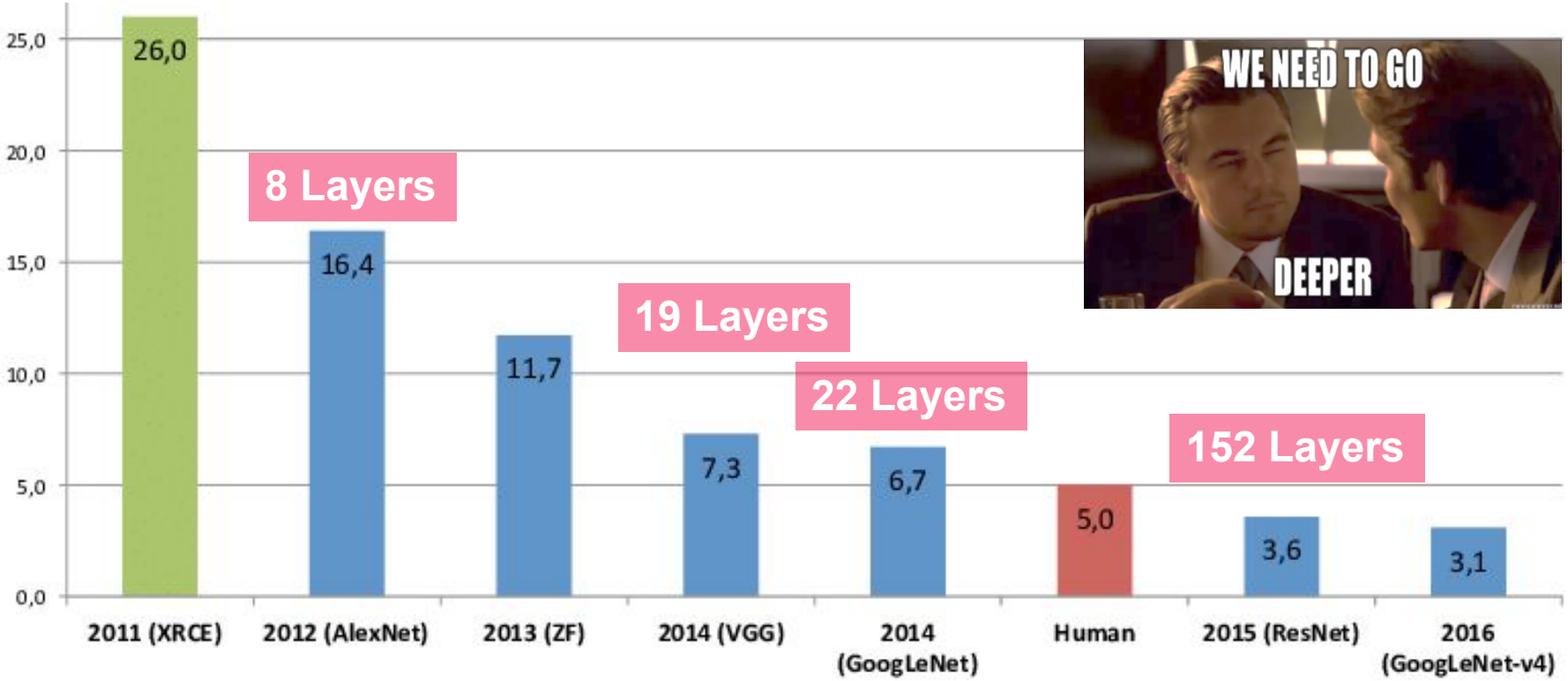
Residual Networks (ResNets)



Residual Networks (ResNets)

- Residual blocks (multiple convolutions with **skip connections**)
- Downsampling using stride 2, instead of max/avg pooling
- Global average pooling after last convolutional layers (introduced by Network-in-Network)
    - Means that the embedding has no spatial dimension and is only 512 floats!
- Only a **single** fully-connected **classification layer**
    - learned embeddings are so good we don't need a complex classifier at end of model

ILSVRC 2011-2016: Classification Error (Top 5)



Post-ILVRC

- The last ILVRC competition was held in 2017
- ResNets are still the most popular architecture for a wide variety of problems
- Object recognition in-the-wild is still not solved

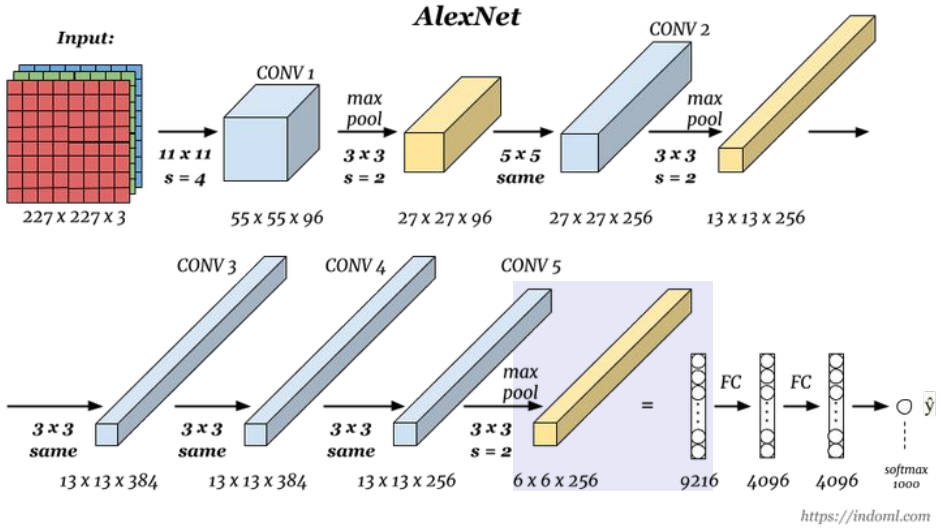
Transfer Learning

Learning Visual Features

Two distinct parts:

- **Convolutional Layers** → Learn filters across spatial and channel dimensions
- **Fully-connected Layers** → Learn to classify images based on the learned visual features

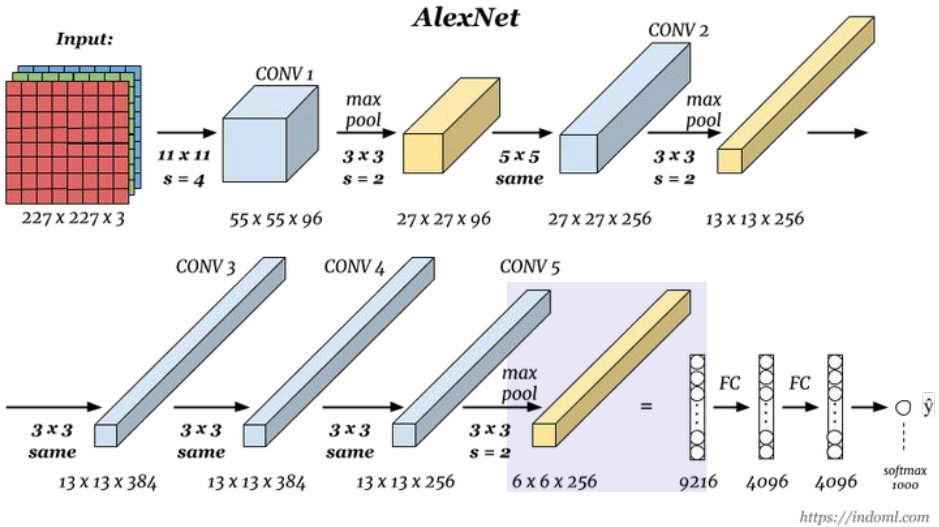
The point at which these parts intersect is an **embedding** → a learned lower-dimensional set of "visual feature" representing the image



This embedding encodes everything needed from the image to classify objects

Transfer Learning using Embeddings

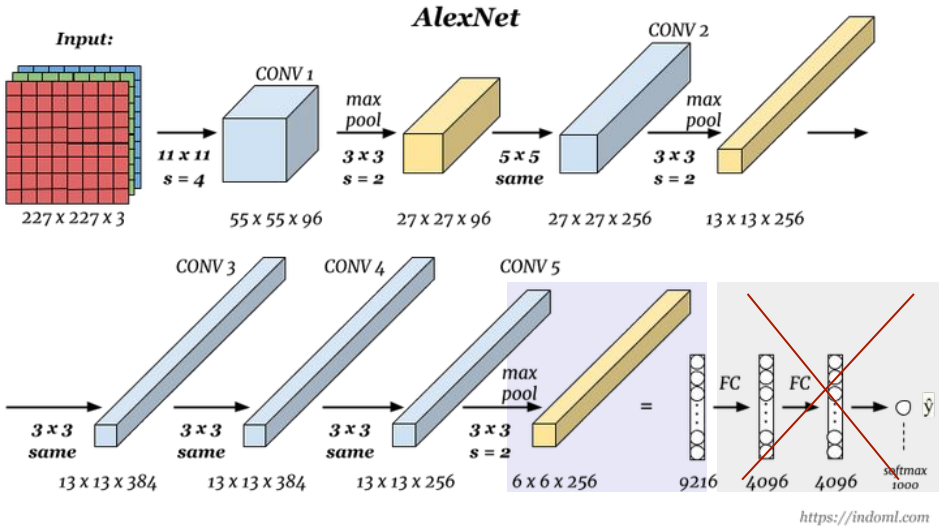
- These CNN embeddings have proven useful for solving a wide variety of image-based problems
- By being trained on a large image classification dataset, **CNNs learn something general about representing images** !



Transfer Learning using Embeddings

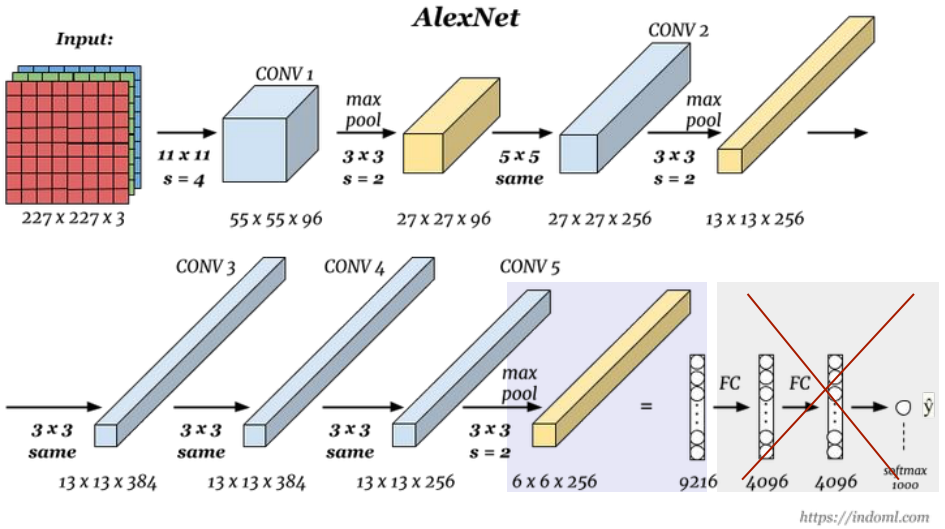
We can use these features to **transfer our learning to a new problem** :

1. Train CNN (e.g. AlexNet) on large image dataset (e.g. ImageNet)
2. Remove "classification" layers at end of model, **freeze remaining weights** .
3. Add, and train, **new layers at end** of model suitable for our new task.



Transfer Learning: Fine-tuning

- We froze the original model's weights, and used our CNN layers as a feature extractor
- Often training some/all of the original model's weights on the new task at a lower learning rate helps the features "adapt" to the new task
- This, and variants of it, is often referred to as **fine-tuning**



Transfer Learning: PyTorch

- All of the models we've discussed (and more!) are available in torchvision
- We can avoid the **large computation** needed for training a state-of-the-art model, and just use pre-trained models
- You can also train the models from scratch with → `pretrained=False`
- Keep this in mind for your projects!

```
import torchvision.models

alexnet= torchvision.models.alexnet(pretrained=True)

Inception=
torchvision.models.inception.inception_v3(pretrained=True)

vgg16= torchvision.models.vgg.vgg16(pretrained=True)
vgg19= torchvision.models.vgg.vgg19(pretrained=True)

resnet18=
torchvision.models.resnet.resnet18(pretrained=True)

resnet152=
torchvision.models.resnet.resnet152(pretrained=True)

feature_data = alexnet(image)
```

Questions?