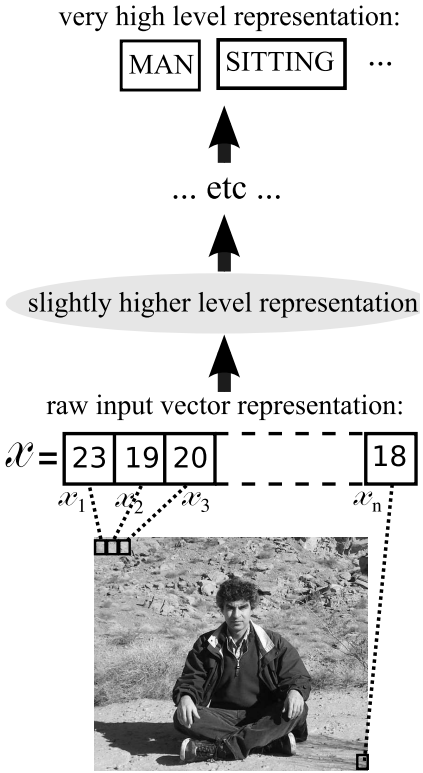
# COMP 551 - Applied Machine Learning Lecture 15 — Convolutional Neural Nets

William L. Hamilton

(with slides and content from Joelle Pineau)

# The deep learning objective



# Major paradigms for deep learning

- Deep neural networks: The model should be interpreted as a computation graph.
  - Supervised training: E.g., feedforward neural networks.
  - Unsupervised training (later in the course): E.g., autoencoders.

- Special architectures for different problem domains.
  - Computer vision => Convolutional neural nets.
  - Text and speech => Recurrent neural nets.

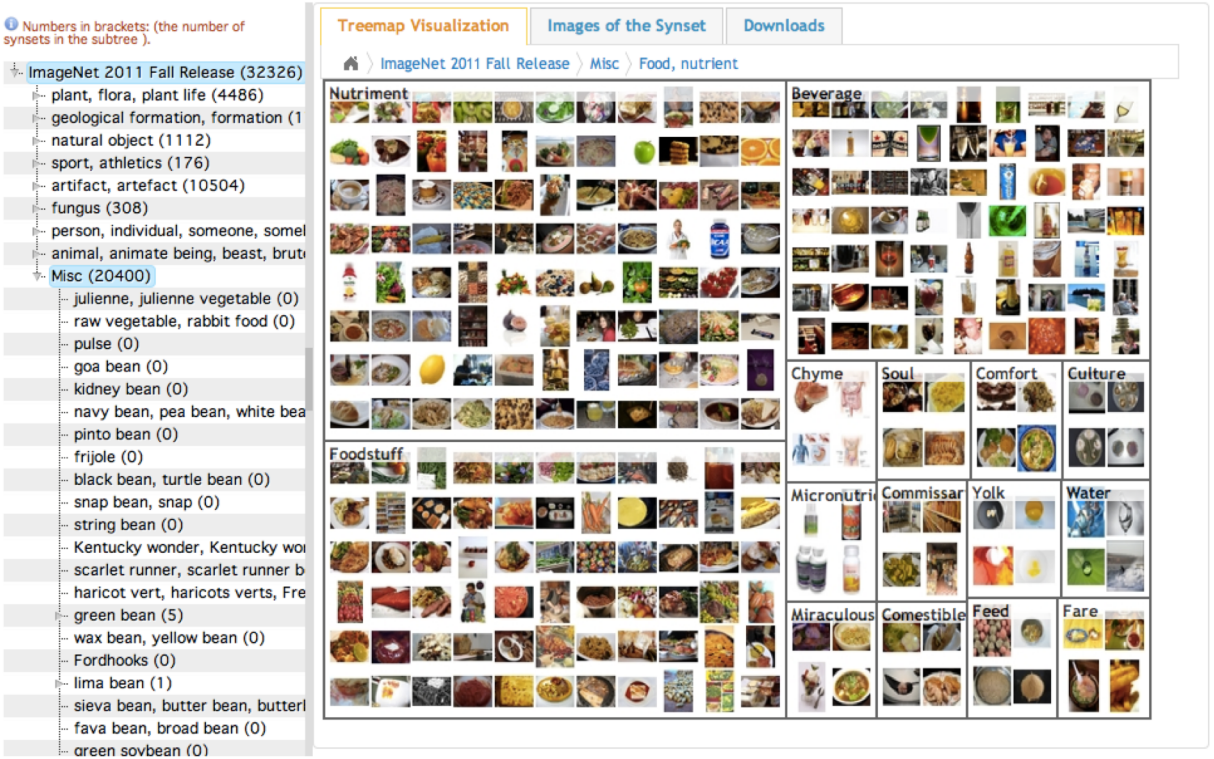
# Major paradigms for deep learning

- Deep neural networks: The model should be interpreted as a computation graph.
  - Supervised training: E.g., feedforward neural networks.
  - Unsupervised training (later in the course): E.g., autoencoders.

- Special architectures for different problem domains.
  - Computer vision => Convolutional neural nets.
  - Text and speech => Recurrent neural nets.

# ImageNet Challenge

- 1.2 million images
- 1 thousand object categories.
- Goal is to classify what type of object is present in an image.



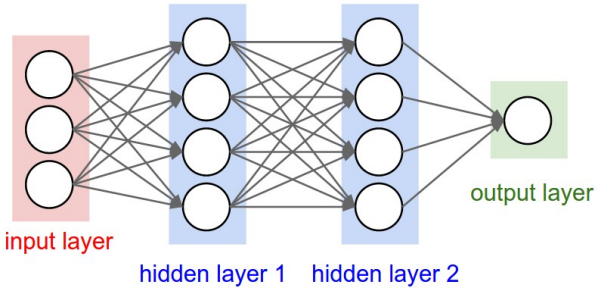

http://www.image-net.org

# Neural networks for computer vision

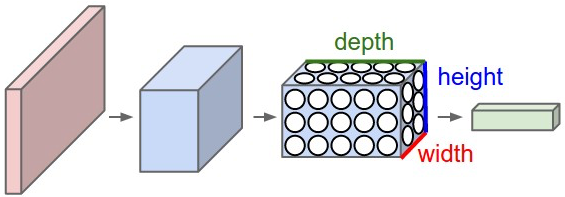
- Design neural networks that are specifically adapted to:
  - Deal with very high-dimensional inputs
    - E.g. 150x150 pixels = 22,500 inputs, or 3x22,500 if RGB
  - Exploit 2D topology of pixels (or 3D for video)
  - Built-in invariance to certain variations we can expect
    - Translations, illumination, etc.

# Convolutional Neural Networks

Feedforward network




Convolutional neural network (CNN)



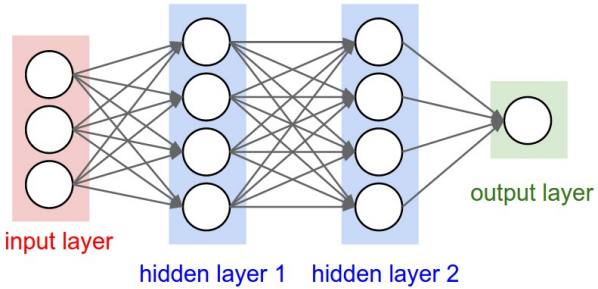

- CNN characteristics:
  - Input is usually a 3D tensor:  2D image x 3 colours
  - Each layer transforms an input 3D tensor to an output 3D tensor using a differentiable function.



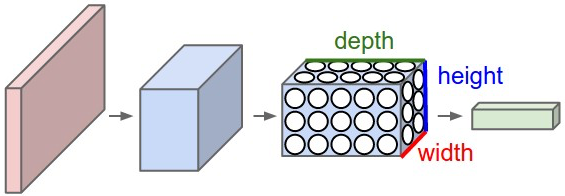
From: *http://cs231n.github.io/convolutional-networks/*

# Convolutional Neural Networks

Feedforward network




Convolutional neural network (CNN)



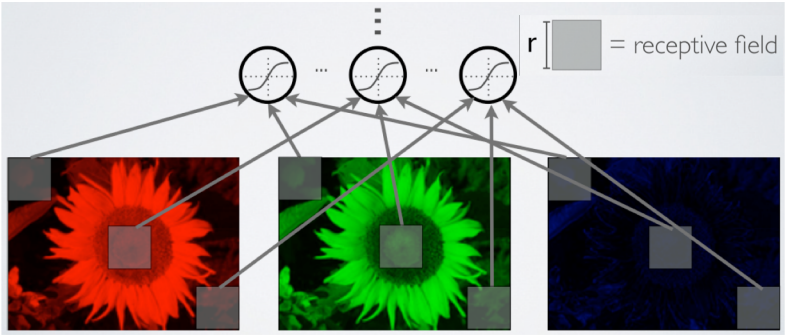

- Convolutional neural networks leverage several ideas.
  1. Local connectivity.
  2. Parameter sharing.
  3. Pooling hidden units.

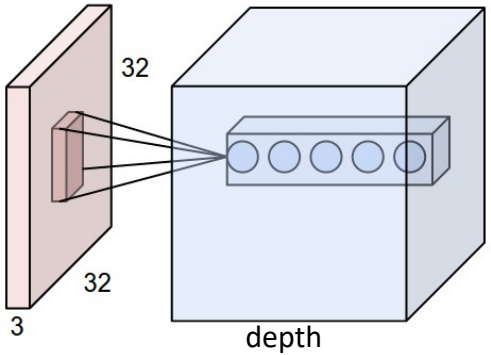


From: *http://cs231n.github.io/convolutional-networks/*

# Convolutional Neural Networks: Key Idea 1

1. Features have local receptive fields.
   - Each hidden unit is connected to a patch of the input image.
   - Units are connected to all 3 colour channels.

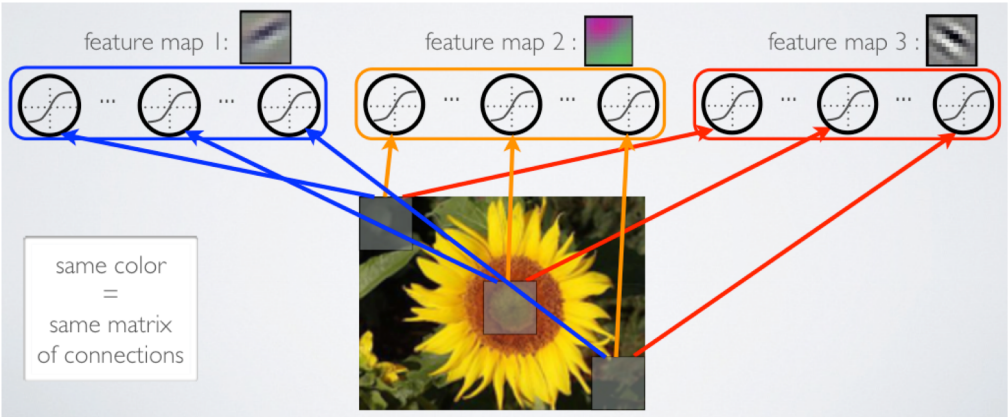



depth = # filters
(a hyperparameter)

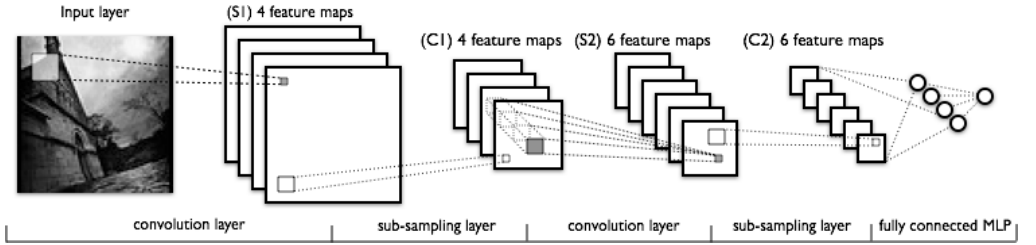
# Convolutional Neural Networks: Key Idea 2

2. Share matrix of parameters across units.
   - Constrain units within a depth slice (at all positions) to have **same** weights.
   - Feature map can be computed via discrete convolution with a kernel matrix.



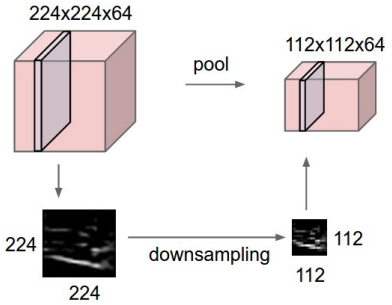
# Convolutional Neural Networks: Key Idea 3

3. Pooling/subsampling of hidden units in same neighbourhood.




Dimensionality is reduced via subsampling/pooling

Features become more high-level and abstract




Example:

Single depth slice

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

x

y

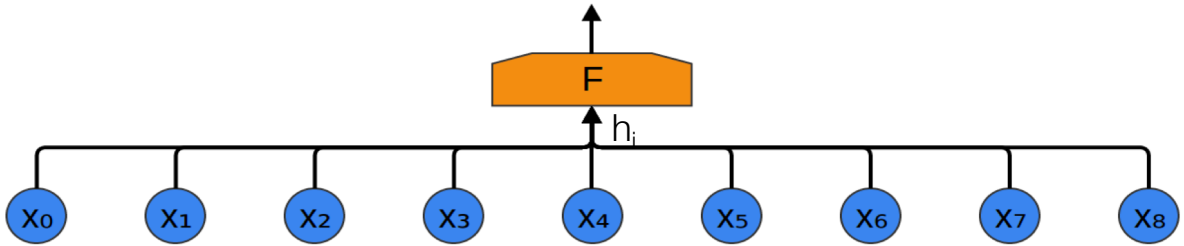
max pool with 2x2 filters and stride 2

| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

From: *http://cs231n.github.io/convolutional-networks/*

# Feed-forward NNs to CNNs: In detail

- Consider 1-D inputs for simplicity, then generalize to 2-D.
- In a standard FF-NN, we feed all the input features to the hidden layer:




$$h_i = \phi(\mathbf{W}\mathbf{x} + b), \mathbf{x} = [x_1, x_2, ..., x_n]$$

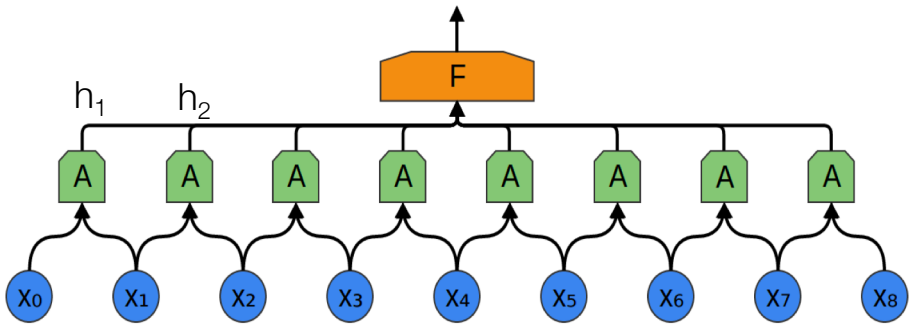
# Feed-forward NNs to CNNs: In detail

- In a convolutional layer, we group the input units together, and apply **the same function** to these different groups.
- This is the idea of **parameter sharing** and **local receptive fields.**

$$h_1 = \phi(\mathbf{W}[x_1, x_2]^\top + b)$$

$$h_2 = \phi(\mathbf{W}[x_2, x_3]^\top + b)$$

The parameters for the hidden units are shared… but the inputs are different and local!



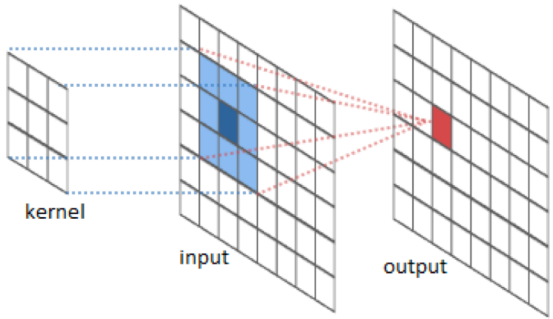
# Feed-forward NNs to CNNs: In detail

- In a convolutional layer, we group the input units together, and apply **the same function** to these different groups.
- This is the idea of **parameter sharing** and **local receptive fields.**
- Mathematically equivalent to a discrete convolution operation.

$$h_1 = \phi(\mathbf{W}[x_1, x_2]^\top + b)$$

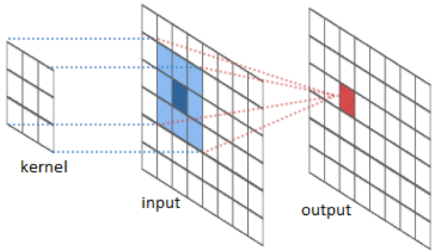
$$h_2 = \phi(\mathbf{W}[x_2, x_3]^\top + b)$$

The parameters for the hidden units are shared… but the inputs are different and local!



# Feed-forward NNs to CNNs: In detail

- The goal is to learn convolutional "kernels" that extract useful information.
- Some examples:




| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1/9 | 1/9 | 1/9 | 0 |
| 0 | 1/9 | 1/9 | 1/9 | 0 |
| 0 | 1/9 | 1/9 | 1/9 | 0 |
| 0 | 0 | 0 | 0 | 0 |

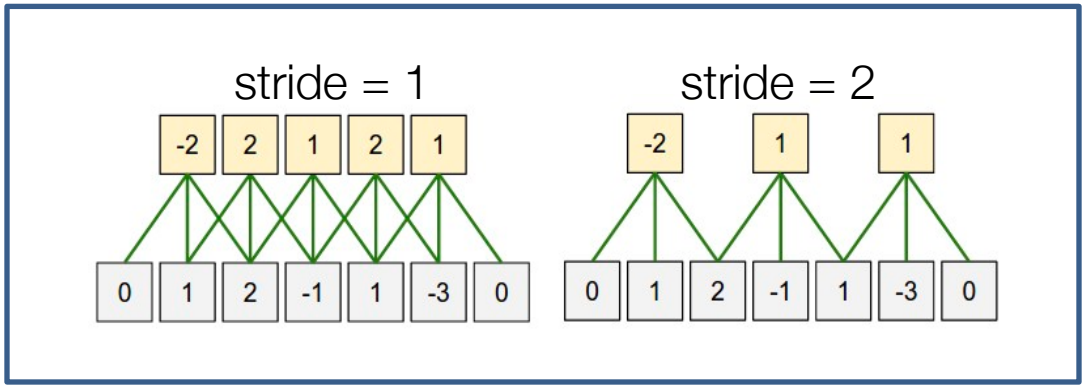
Blurring, i.e., averaging pixels in the receptive field

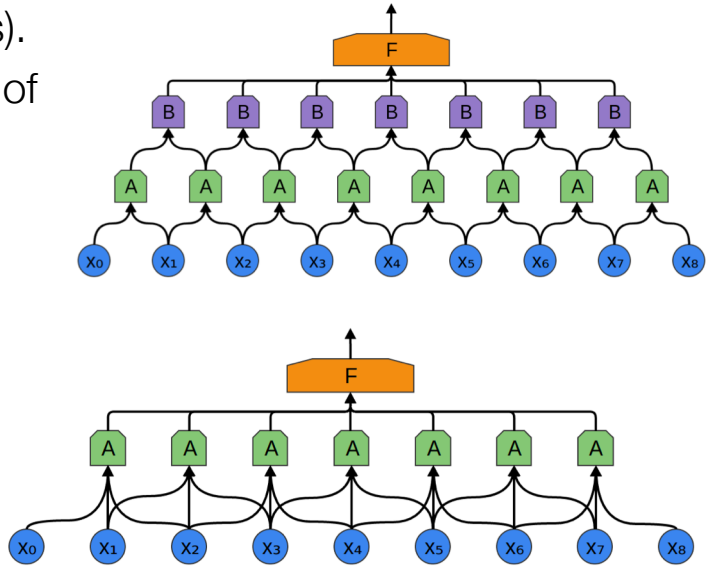
| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Vertical edge detector, i.e., subtracting side-by-side pixels

# Feed-forward NNs to CNNs: In detail

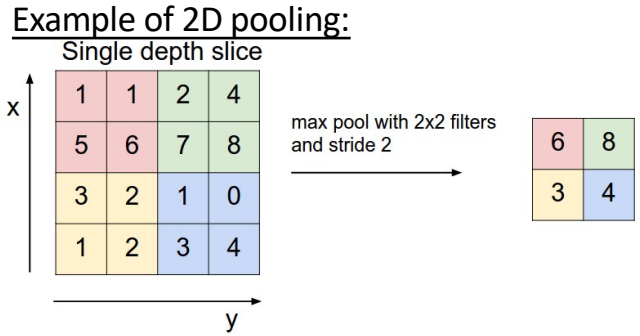
- Can stack multiple convolutional layers.
- Can vary the width/size of the receptive field.
- Can apply multiple convolutions to the same input.
- Can vary the "stride" (i.e., spacing between receptive fields).
- It is common to add zero-padding to allow the application of the kernel near the boundary.

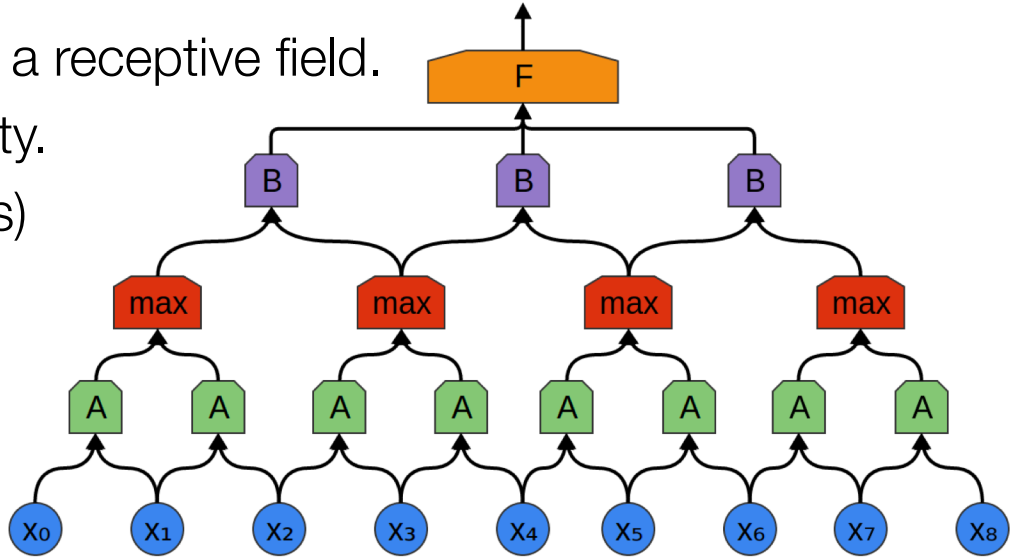


# Feed-forward NNs to CNNs: In detail

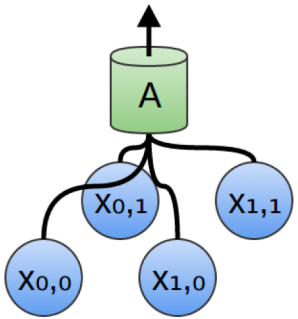
- Pooling layers are often inserted between convolutional layers.
- Simply take the  max of inputs in a receptive field.
- Further reduces the dimensionality.
- (Less popular in last couple years)



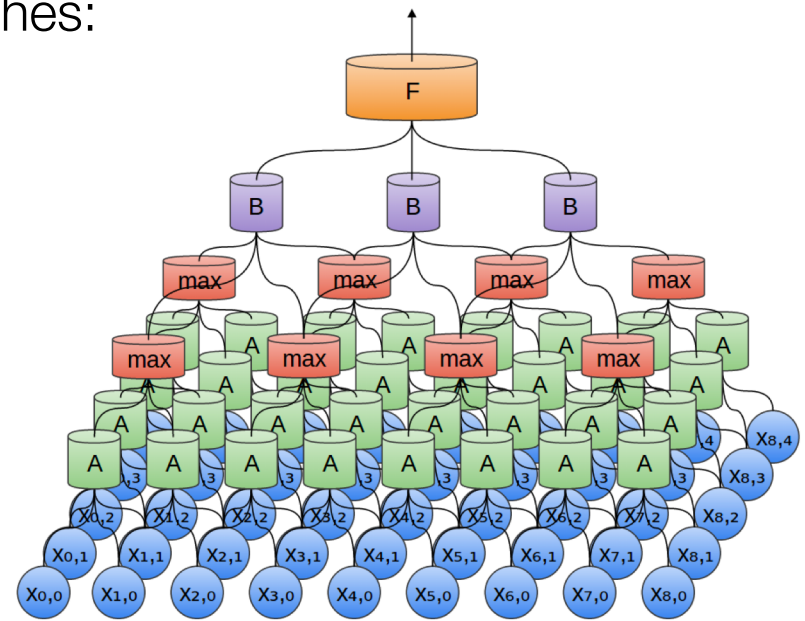



# Feed-forward NNs to CNNs: In detail

- In 2D we group patches together in receptive fields.
- I.e., a 2x2 convolution groups 2x2 patches:

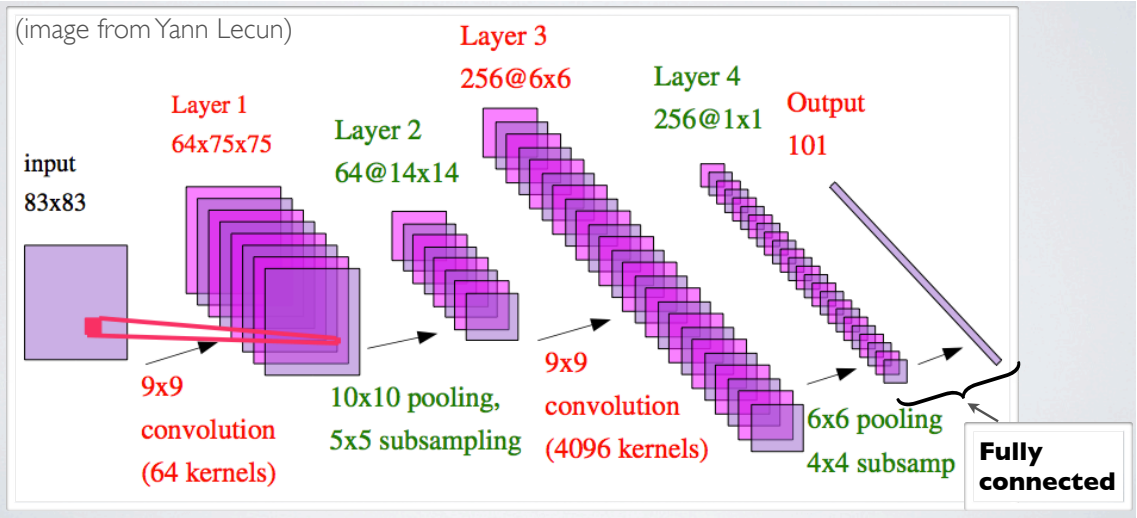



$$h_1 = \phi(\mathbf{W}[x_{0,0}, x_{0,1}, x_{1,1}, x_{1,1}]^\top + b)$$



# CNNs: Putting it all together

- Alternate between convolutional, pooling, and fully connected layers.
  - Fully connected layer typically only at the end.
- Train full network using backpropagation.

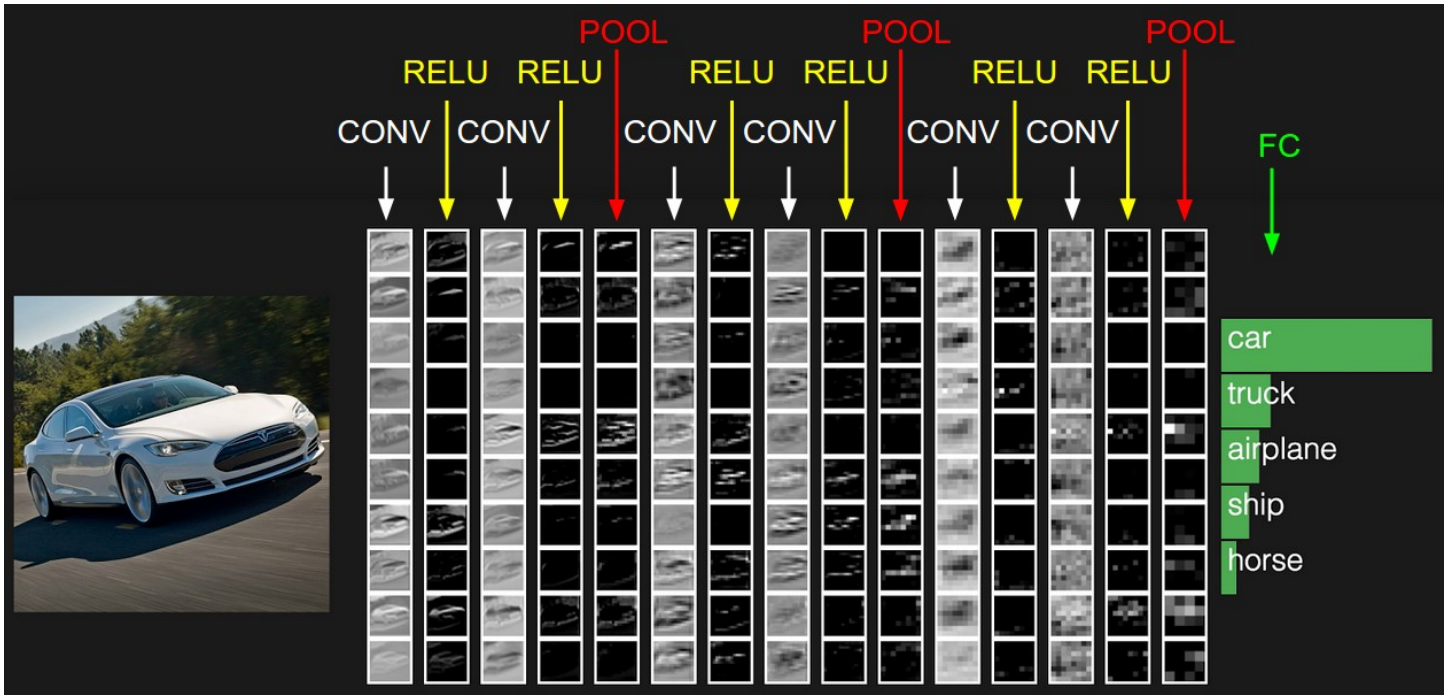


# CNNs: Example in PyTorch

```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

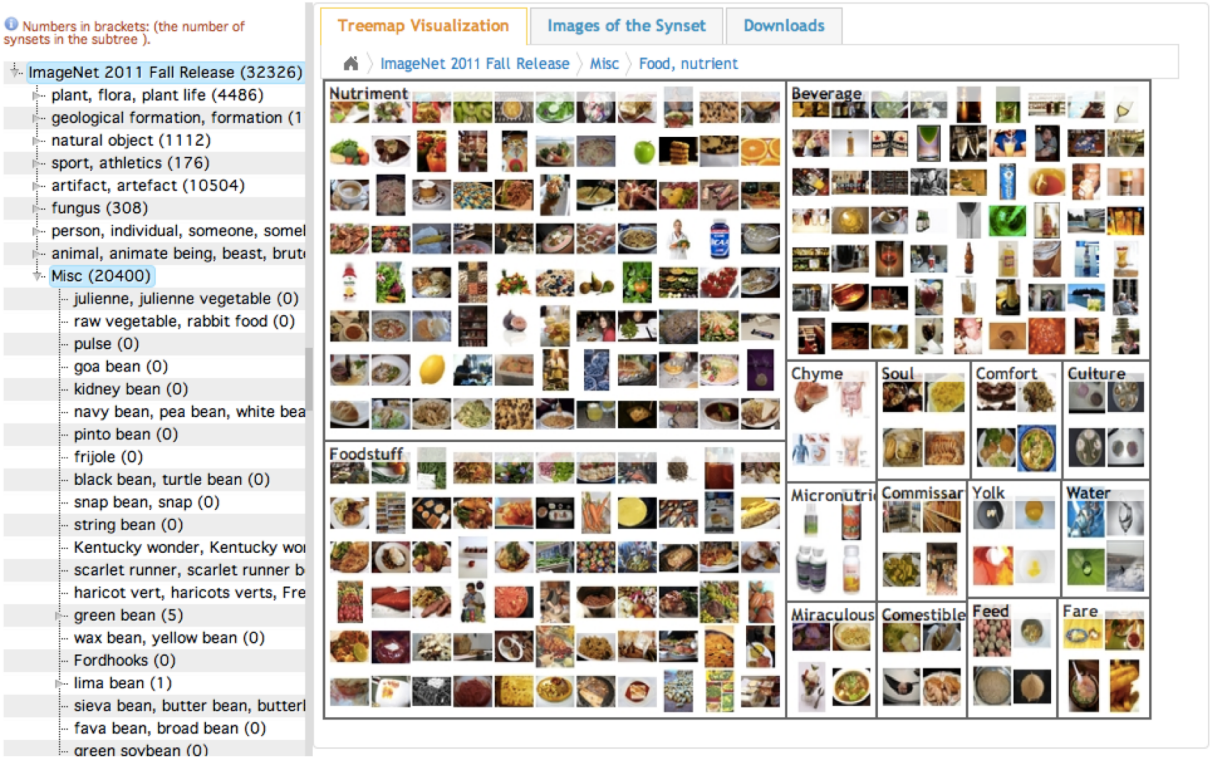
# CNNs: Example




From: *http://cs231n.github.io/convolutional-networks/*

# ImageNet Challenge

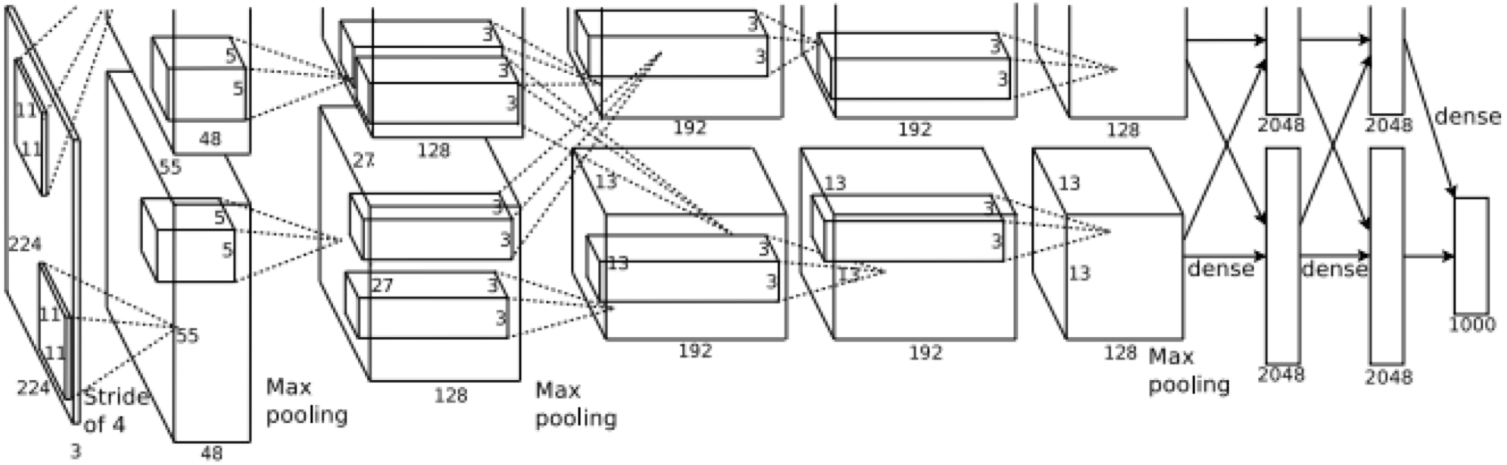
- Millions of images.
- Thousands of object categories.
- Goal is to classify what object is present in an image.




http://www.image-net.org

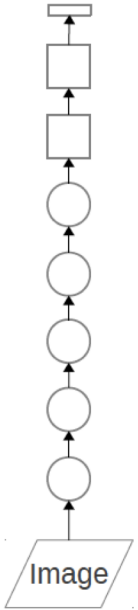
# Example: ImageNet with AlexNet

- SuperVision (a.k.a. AlexNet, 2012):



# Example: ImageNet with AlexNet

- SuperVision (a.k.a. AlexNet, 2012):



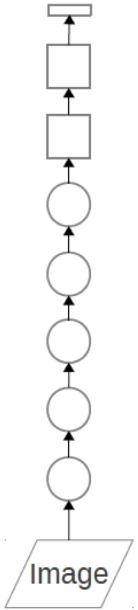

- Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- **Final feature layer:** 4096-dimensional

**Convolutional layer:** convolves its input with a bank of 3D filters, then applies point-wise non-linearity

**Fully-connected layer:** applies linear filters to its input, then applies point-wise non-linearity

# Example: ImageNet with AlexNet

- SuperVision (a.k.a. AlexNet, 2012):

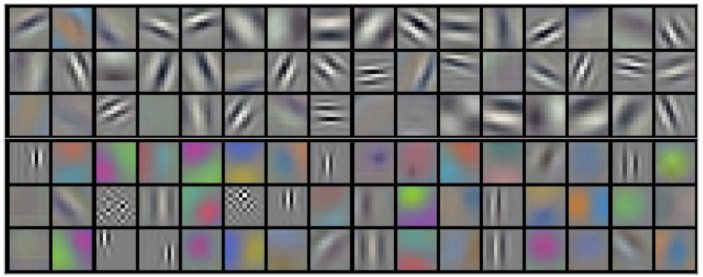



- Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- **Final feature layer:** 4096-dimensional

**Convolutional layer:** convolves its input with a bank of 3D filters, then applies point-wise non-linearity

**Fully-connected layer:** applies linear filters to its input, then applies point-wise non-linearity

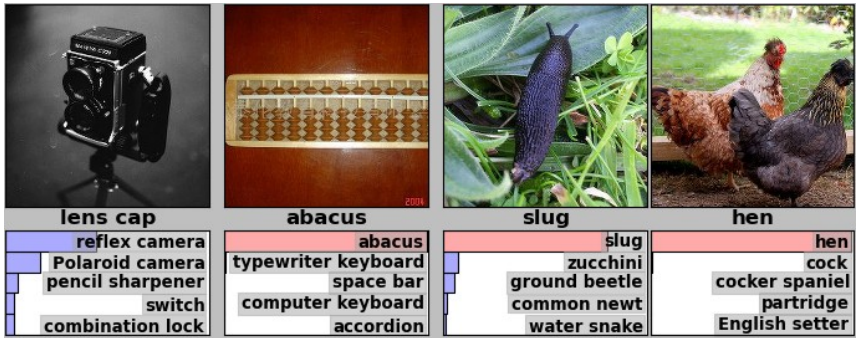
Example features learned by the first convolutional layers:



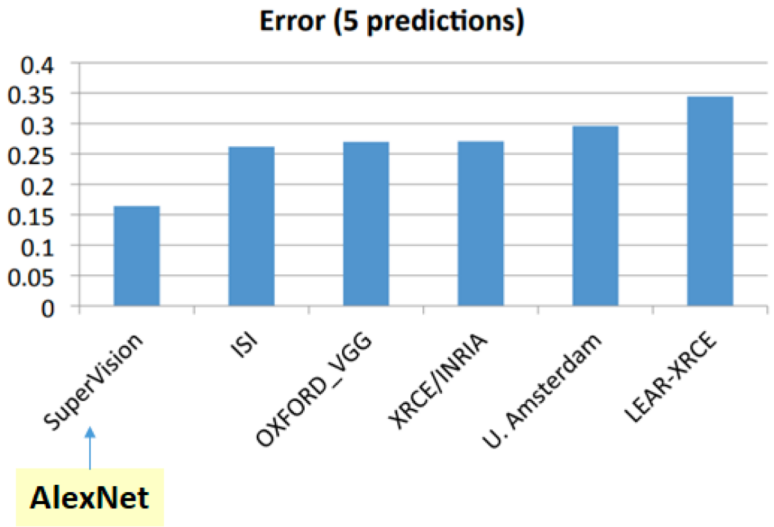
# ImageNet results (2012)

- AlexNet drastically outperformed the competition.
- AlexNet success marked the start of the “deep learning” era of machine learning.

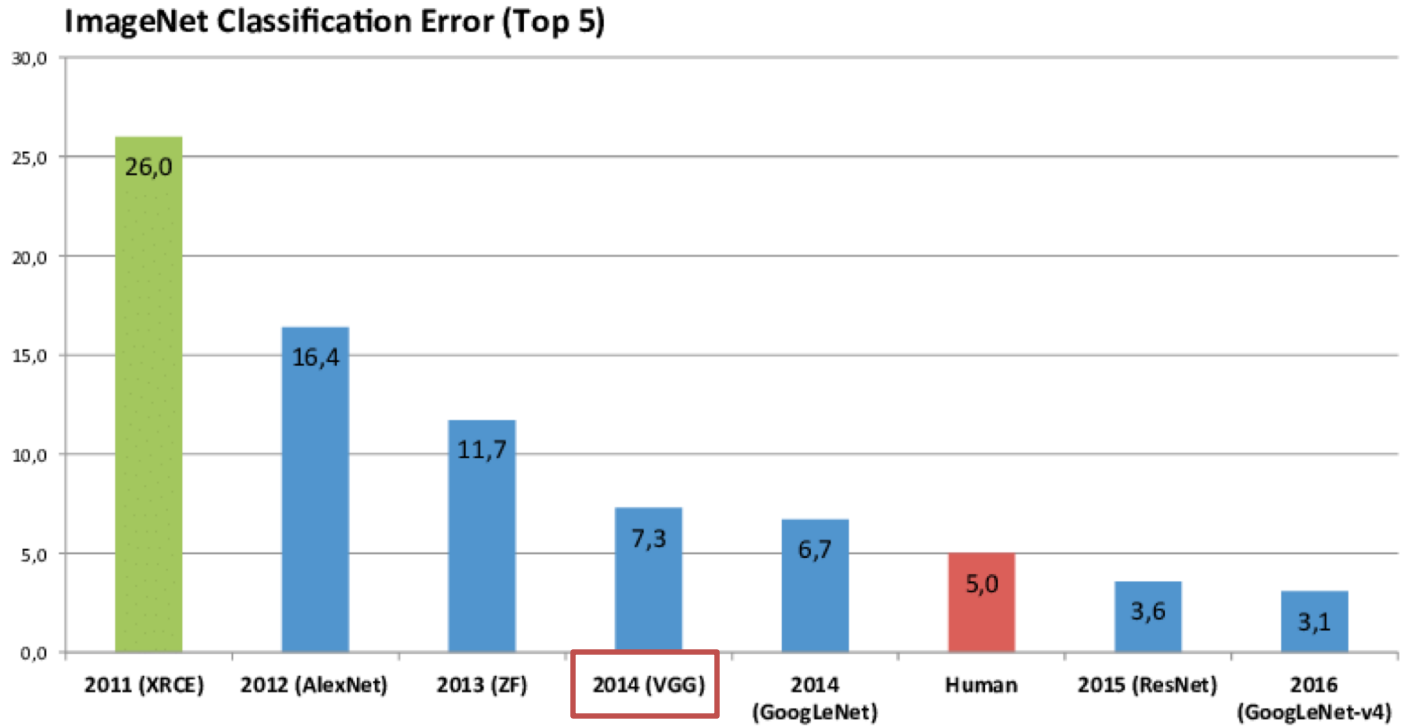
Example AlexNet predictions:




Ranking of the best results from each team

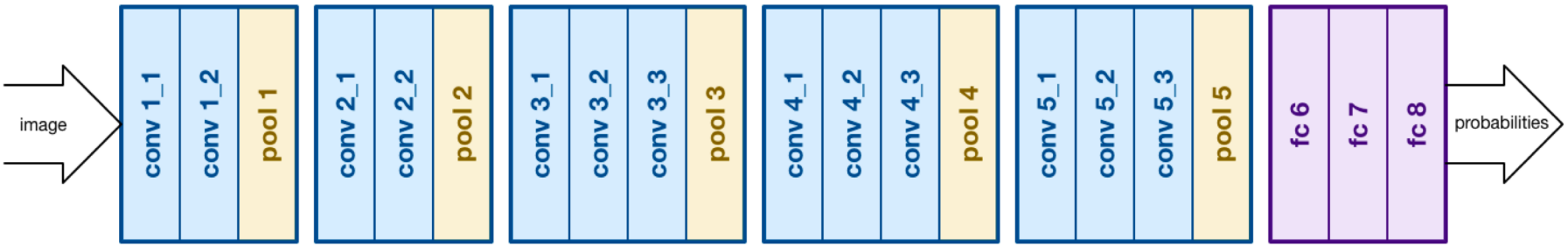


# ImageNet results (2016)

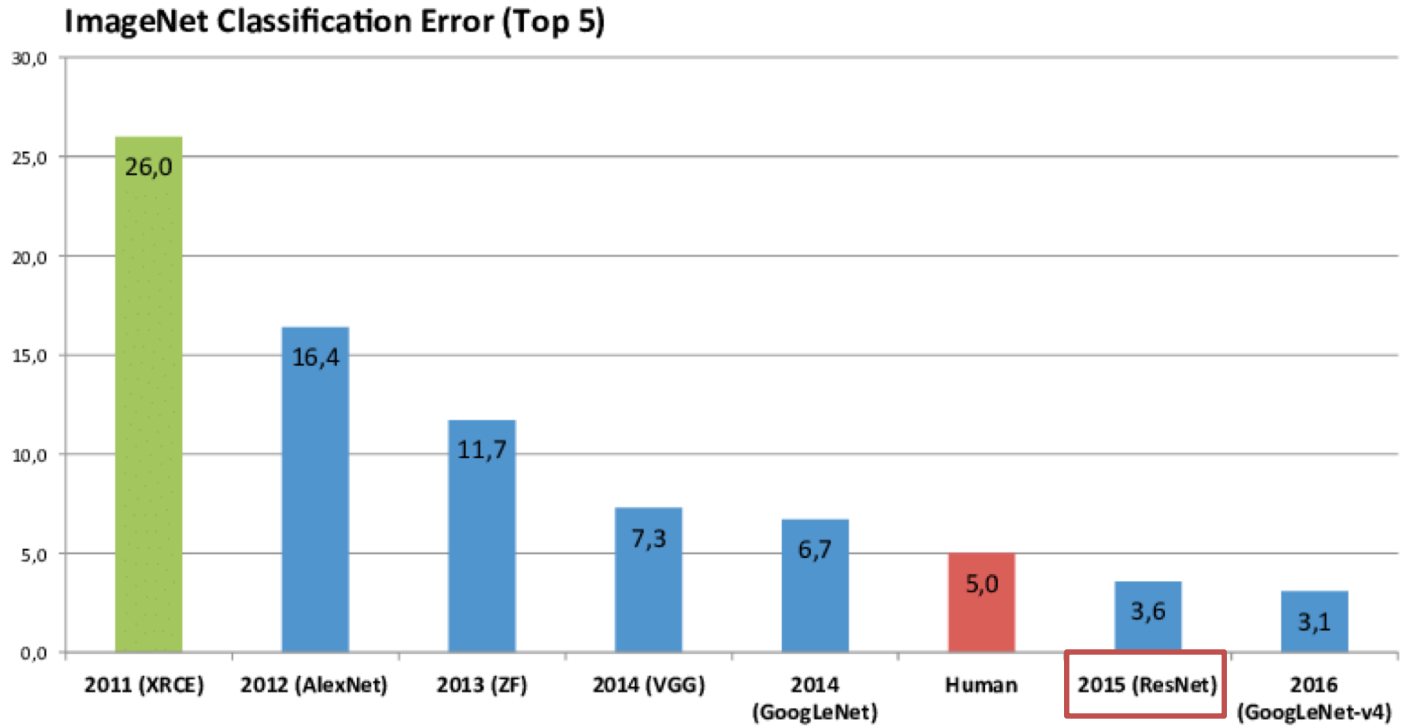


# VGG Net

- 16 layers (not including pooling)
- 3x3 convolutions only
- 138 million parameters
- Pre-trained model is available for download and is a standard in the field

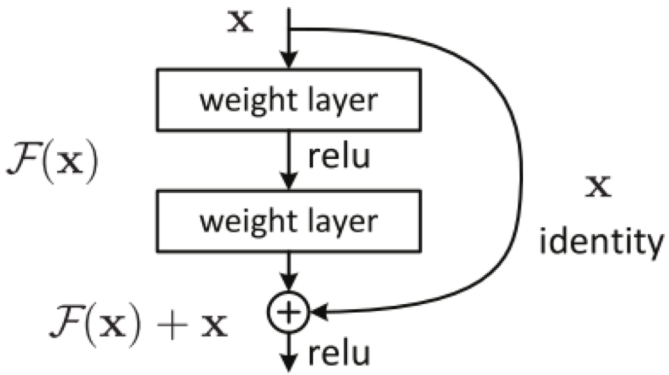


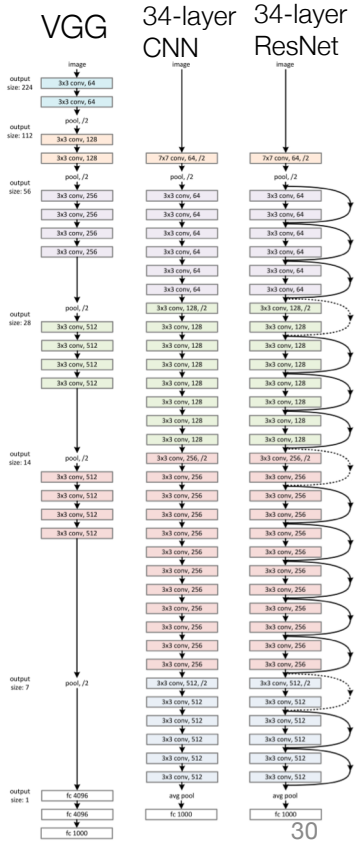
# ImageNet results (2016)



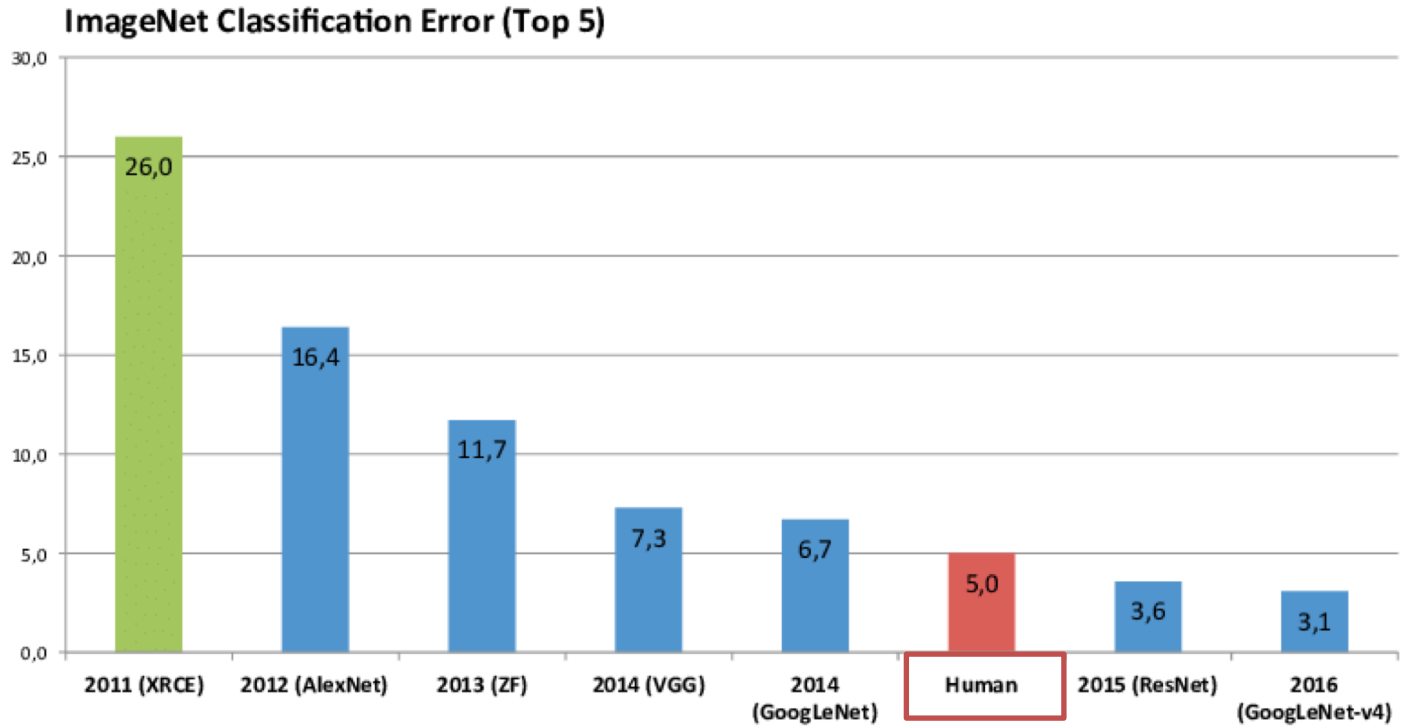
# Residual networks (ResNets)

- Simple idea: Add “skip” or “residual” connections between layers.
- Allows for much, much deeper models with hundreds or even thousands of layers!
- Key component of most state-of-the-art CNNs.



# ImageNet results (2016)

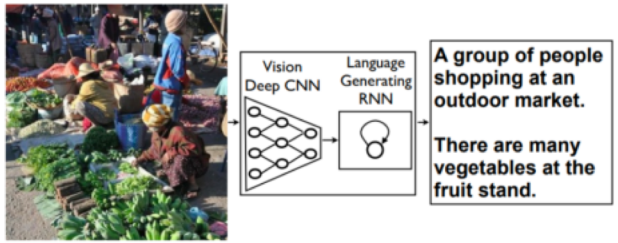


# Achieving super-human performance?

- Estimated 3% error in the labels.
- Differences between labeling process and human assessment:
  - Labels acquired as binary task. *Is there a dog in this picture?*
  - Human performance measured on 1K classes (>120 species of dogs in the dataset).
  - Labels acquired from experts (dog experts label the dogs, etc.).
- Machines and humans make different kinds of mistakes.
  - Both have trouble with multiple objects in an image.
  - Machines struggle with small/thin objects, image filters.
  - Humans struggle with fine-grained recognition.

*http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/*

# Advanced CNN applications: Image captioning





"girl in pink dress is jumping in air."


"black and white dog jumps over bar."


"young girl in pink shirt is swinging on swing."


"man in blue wetsuit is surfing on wave."


"little girl is eating piece of cake."


"baseball player is throwing ball in game."


"woman is holding bunch of bananas."


"black cat is sitting on top of suitcase."

# Advanced CNN applications: Scene parsing

# Practical tips for CNNs

- Many hyper-parameters to choose!
- **Architecture**: filters (start small, e.g. 3x3, 5x5), pooling, number of layers (start small, add more).
- Read papers, copy their method, then do local search.
- Consider starting with VGGNet!
- Consider using dropout and/or BatchNorm.

# Tip #1: Dropout regularization

- **Goal**: Learn model that generalizes well, robust to variability.
- **Method**: Independently set each hidden unit activity to zero with probability $p$ (usually $p$=0.5 works best).
- **Effect**: Can greatly reduce overfitting.

# Tip #2: Batch normalization

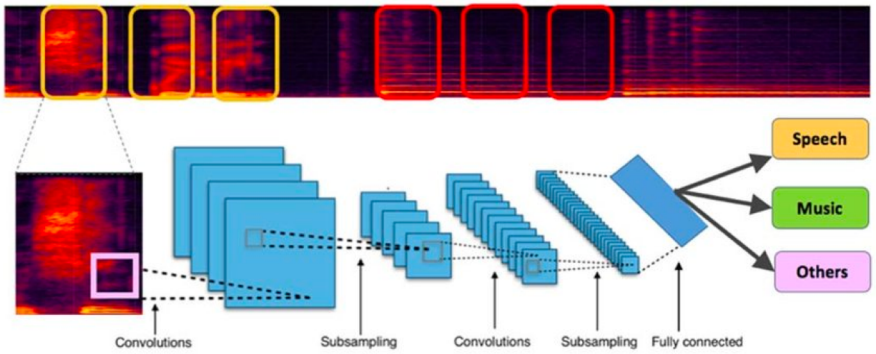
- Idea: Feature scaling/normalization makes gradient descent easier.
  - We already apply this at the input layer; extend to other layers.
  - Use empirical batch statistics to choose re-scaling parameters.

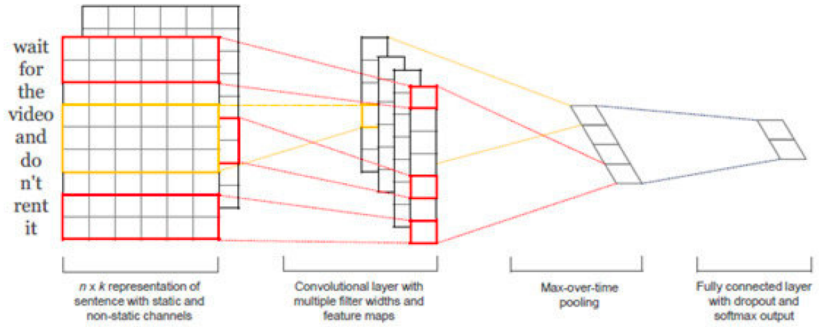
- For each mini-batch of data, at each layer *k* of the network:
  - Compute empirical mean and var independently for each dimension
  - Normalize each input: $\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{VAR[x^{(k)}]}}$
  - Output has tunable parameters ($\gamma$,$\beta$) for each layer: $y^k = \gamma^k . \hat{x}^{(k)} + \beta^k$

- Effect: More stable gradient estimates, especially for deep networks.

# CNNs beyond vision

- CNNs are not only useful for image tasks!
- They are becoming the standard in audio tasks and very competitive in text processing tasks (e.g., sentiment classification).



# Quick recap + more resources

- A good survey paper:
  - Bengio, Courville, Vincent. Representation learning: A Review and New Perspectives. IEEE T-PAMI. 2013. *http://arxiv.org/pdf/1206.5538v2.pdf*

- Notes and images in today's slides taken from:
  - *http://cs231n.github.io/convolutional-networks/*
  - *https://colah.github.io/posts/2014-07-Conv-Nets-Modular/*
  - *http://www.cs.toronto.edu/~hinton/csc2535*
  - *http://deeplearning.net/tutorial/*
  - *http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf*

# Practical tips: Choosing the learning rate

- Backprop is very sensitive to the choice of learning rate.
  - Too large ⇒ divergence.
  - Too small ⇒ VERY slow learning.
  - The learning rate also influences the ability to escape local optima.

- The learning rate is a critical hyperparameter.

# Adaptive optimization algorithms

- It is now standard to use "adaptive" optimization algorithms.

- These approaches modify the learning rate adaptively depending on how the training is progressing.

- Adam, RMSProp, and AdaGrad are popular approaches, with Adam being the de facto standard in deep learning.
  - All of these approaches scale the learning rate for each parameter based on statistics of the history of gradient updates for that parameter.
  - **Intuition:** increase the update strength for parameters that have had smaller updates in the past.

# Adding momentum

- At each iteration of gradient descent, we are computing an update based based on the derivative of the current (mini)batch of training examples:

Update for weight vector

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}}$$

Derivative of error w.r.t. weights for current minibatch

$$\mathbf{w} \leftarrow \Delta_i \mathbf{w}$$

# Adding momentum

- On i'th gradient descent update, instead of:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}}$$

We do:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}} + \beta \Delta_{i-1} \mathbf{w}$$

The second term is called momentum

# Adding momentum

- On i'th gradient descent update, instead of:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}}$$

We do:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}} + \beta \Delta_{i-1} \mathbf{w}$$

The second term is called momentum

Advantages:

- Easy to pass small local minima.
- Keeps the weights moving in areas where the error is flat.

Disadvantages:

- With too much momentum, it can get out of a global maximum!
- One more parameter to tune, and more chances of divergence