Deep Learning

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org

Image Classification

- Image classification can be a difficult task
- Some of the challenges we have to face are:
  - Viewpoint variation: an object can be oriented in many ways
  - Scale varition: objects can vary in size
  - Deformation: some objects can be deformed
  - Occlusion: only a part of the object is visible
  - Illumination conditions: lighting conditions can vary on an object
  - Background clutter: object may blend into a cluttered background
  - Intra-class variation: categories can be very broad, such as chair

Image Classification

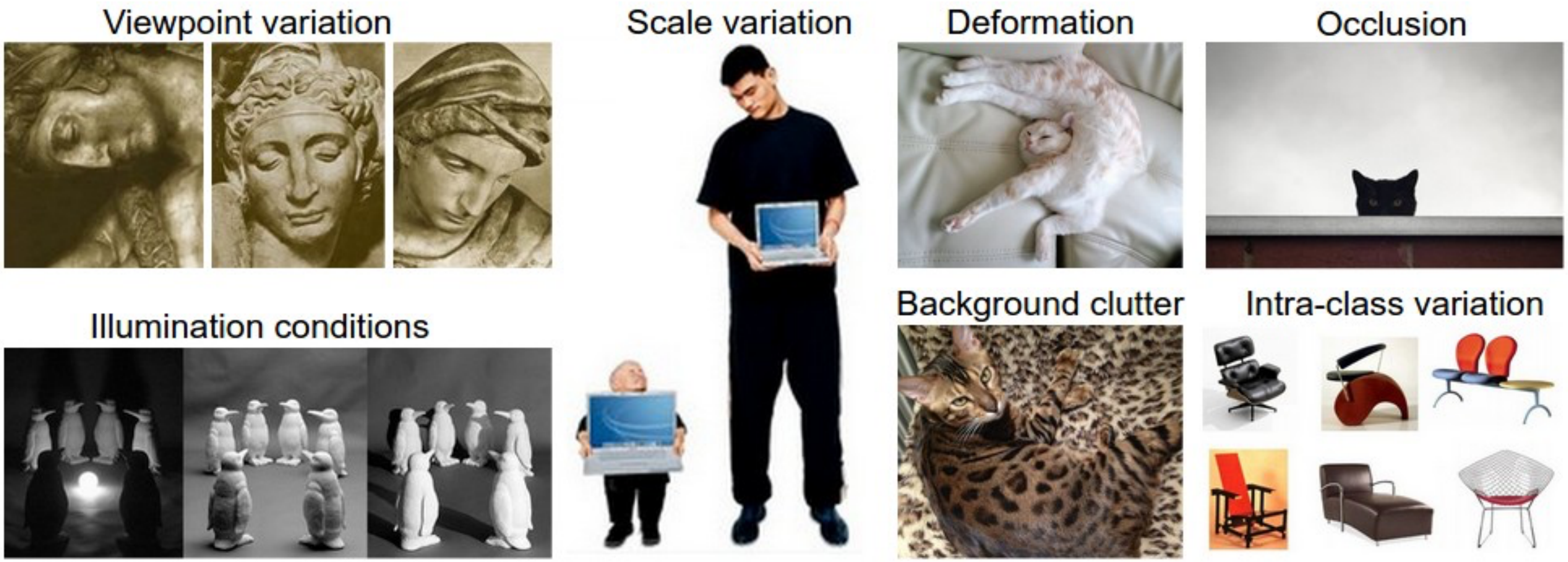


Image Classification

- The dataset can also be very large with lots of categories:

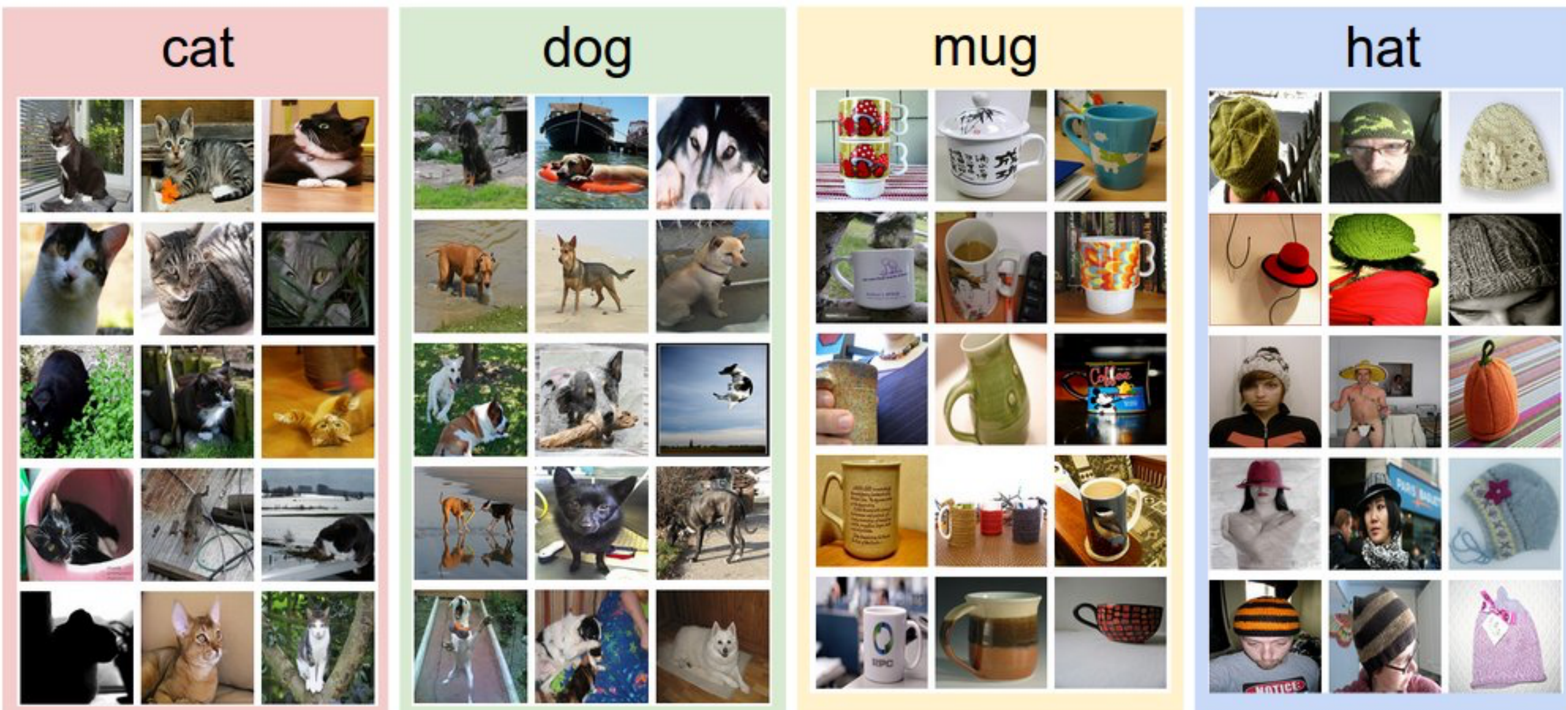
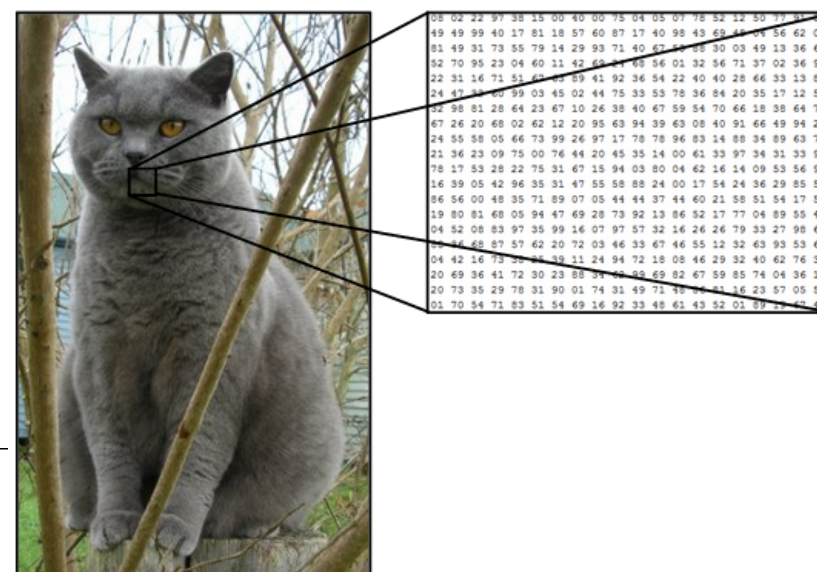


Image Classification

- Each image also requires a lot of input values:
  - Suppose we have an image of 248x400 pixels
  - If the image is in color, we have one value Red, one for Green, and one for Blue (RGB, 3 color channels)
  - The image is made up of 248x400x3 values = 297600 values!



Deep Learning

Deep Learning

- Deep Learning means any deep neural network with more than one hidden layer
- When we talk about deep learning, we often mean specialized deep networks
- The most well known specialized DNN is the Convolutional Neural Network
- This is what we shall focus on in this lecture

ConvNets (CNNs)

- ConvNets are very similar to traditional neural networks:
  - They are made up of units that have learnable weights and biases
  - Each unit performs a dot-product of the weights and inputs, and possible ends with a non-linearity (such as the ReLU function)
  - The output layer maps inputs to a category
  - They have a loss function (such as Softmax)
- So, what are the actual differences?

ConvNets

- ConvNets are only used if the input is images!
- This allows us to specialize the architecture for images
- This makes the score function more efficient and reduces the number of weights in the network

Regular NNs

- In regular NNs, the input is a vector which is transformed through one ore more hidden layers
- Each layer is made up of units, and each unit is fully connected to all units in the previous layer
- Each unit in a layer is independent of the other units in the layer
- The last output layer maps inputs to categories

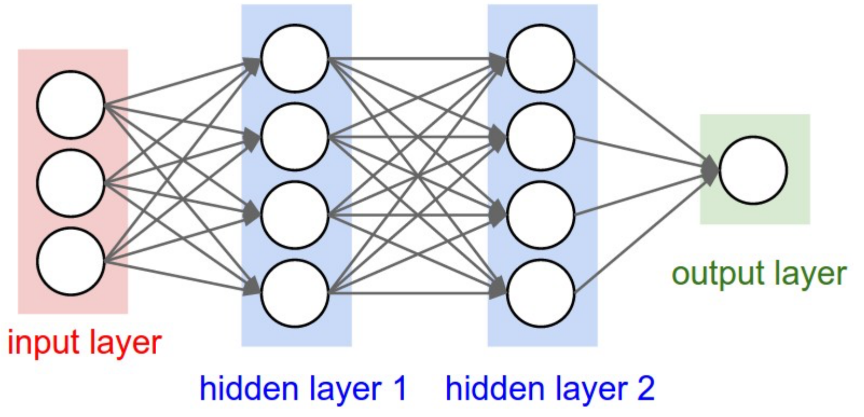
Regular NNs

- Regular NNs don’t scale well to images
- In the CIFAR-10 dataset, each image is 32x32 pixels in 3 color channels
- A fully connected unit would then have 3072 weights
- Since the image recognition task is rather complex, we would need a lot of units!
- If we have larger images, 200x200 pixels, each unit would need 120000 weights!
- Learning all these weights would take a very long time!

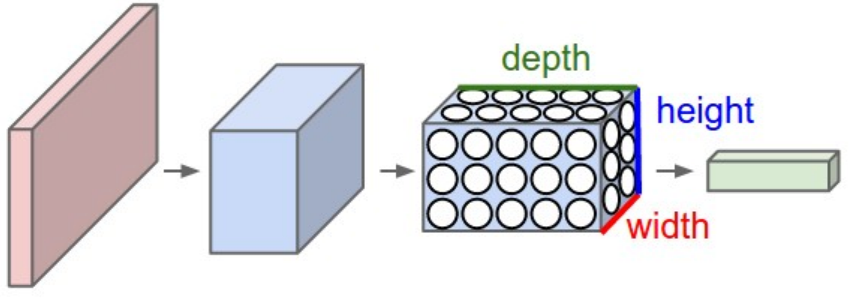
ConvNets

- Images are 3-dimensional: width, height and depth (color channels)
- Each layer in a ConvNet therefore arranges the units in 3 dimensions
- Each unit is also only connected to a small region in the previous layer (not fully connected)
- Each layer transforms the 3D input volume to a new 3D output volume

ConvNets



Regular 3-layer network



3-layer ConvNet

ConvNets

- A ConvNet is a sequence of layers, where each layer transforms one 3D volume to another 3D volume through some function
- There are three main types of layers to use:
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer (identical to regular NNs)
- A sequence of these layers forms a ConvNet architecture

Convolutional Layer

- The Conv layer is the core block of ConvNets
- The Conv layer consist of a set of learnable filters
- Each filter is small along width and height but extends through the full depth of the volume
- A typical filter in the first ConvNet layer can for example have filters of 5x5x3 pixels
- During the forward pass, each filters slides across the width and height of the input volume
- Dot products are computed between each filter and the input volume at any position

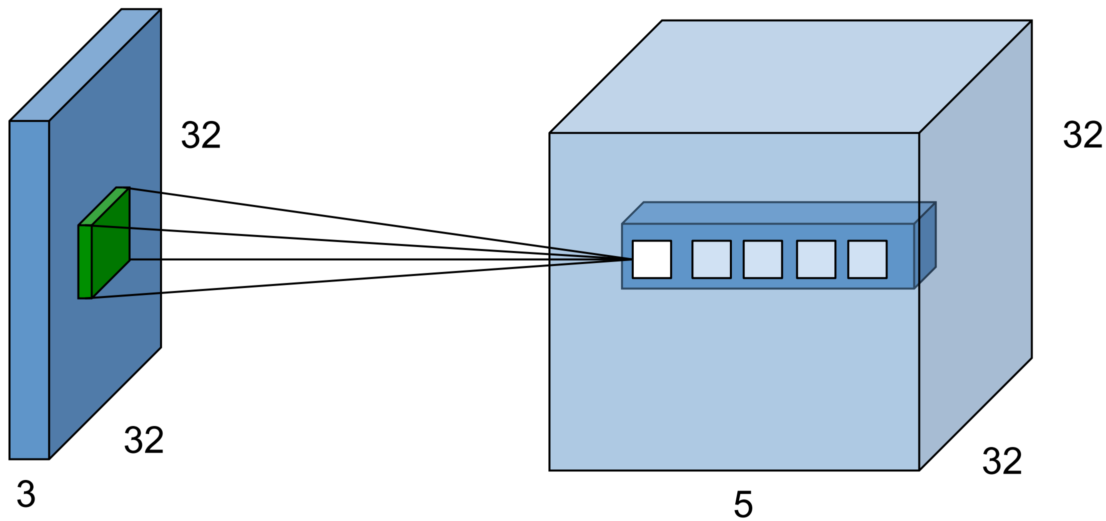
Convolutional Layer

- As the filter slides over the width and height of the input volume, a 2-dimensional activation map is produced
- It gives the response for the current filter at every spatial position in the input volume
- The network will learn filters that activate when they see some interesting visual feature such as an edge, specific color, or more high-level features in later Conv layers
- The Conv layer will have a set of filters (for example 12), and each filter produces a separate 2D activation map
- The activation maps are stacked along the depth dimension and produces the output volume

Convolutional Layer

- Each unit is only connected to a local region of the input volume
- This is referred to as the receptive field of the unit
- Example:
  - We have CIFAR-10 images as input: 32x32x3 pixels
  - The receptive field is 5x5
  - Each unit will then have 5x5x3 weights = 75 weights (and 1 bias)
  - This is much less than 3072 weights needed for a fully connected unit

Convolutional Layer

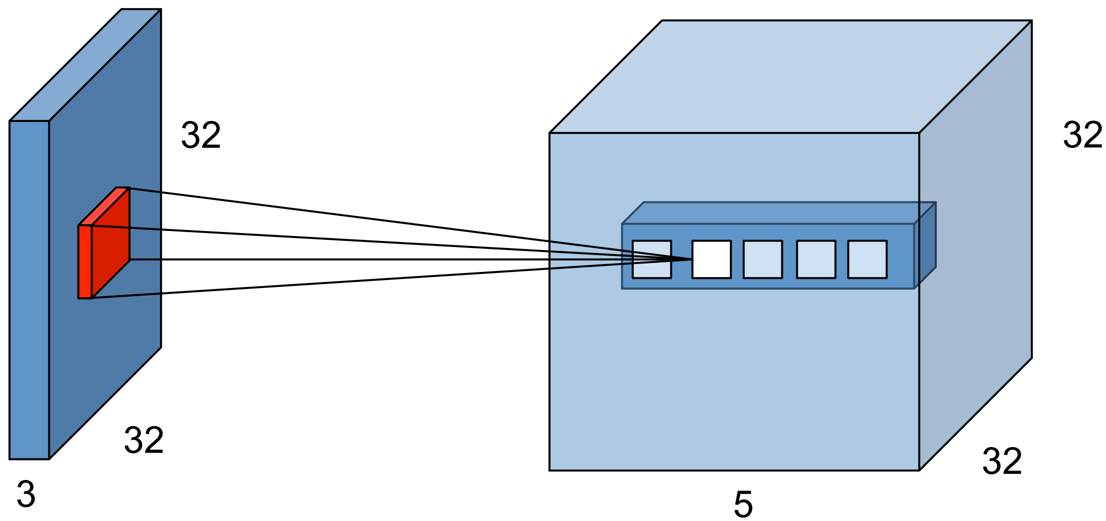


Each 5x5x3 filter slides over every pixel in the input volume

5 filters is used (output volume has depth 5)

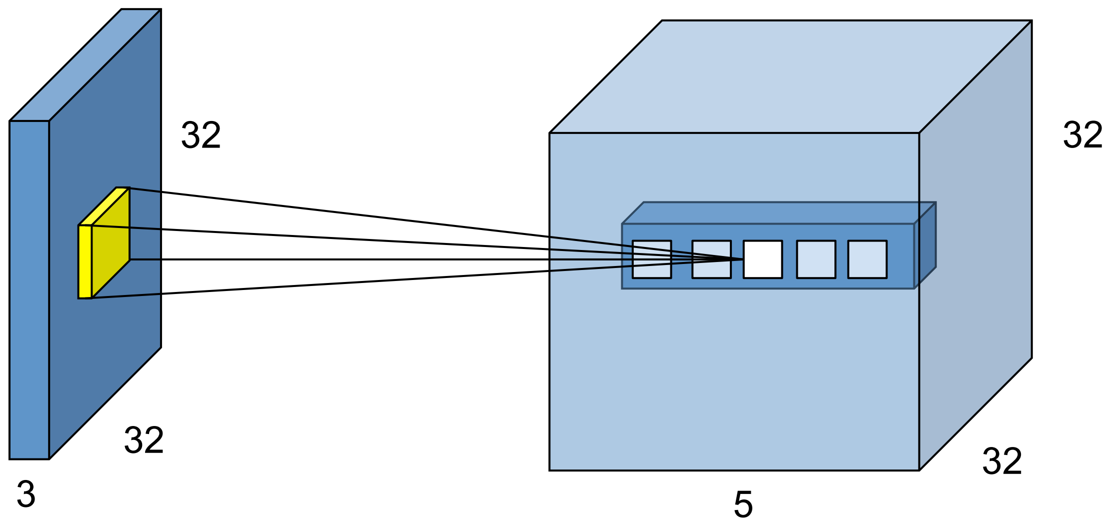
Each filter produces 32x32 values

Convolutional Layer



Second filter slides over
the input volume

Convolutional Layer



Third filter slides over
the input volume

Hyperparameters

- The Conv layer has three hyperparameters: depth, stride and zero-padding
- Depth:
  - The depth of the output volume corresponds to the number of filters we have
- Stride:
  - Stride means how we slide each filter over the input volume
  - In stride 1, the filter is moved one pixel at a time (covering all pixels in the input volume)
  - In stride 2, we jump 2 pixels (covering half of the pixels in the input volume)

Hyperparameters

- Zero-padding:
  - Along the borders of the input volume, some pixels in the volume will be outside the input volume
  - When zero-padding is used, we pad the input volume with zeros around the border to avoid the out-of-bounds issue
  - The parameter determines the size of the zero-padding
  - The size shall be half the filter size for the filters to cover all pixels in the input volume

0	0	0	0	0	0
0	0	0	0	0	0
0	0	45	76	77	83
0	0	53	83	87	92
0	0	55	86	90	95
0	0	56	85	89	95

A 5x5 filter slides over a volume with zero-padding 2

Output volume

- The size of the output volume is determined by:
  - The input volume size, W
  - The receptive field size, F
  - The stride, S
  - The zero-padding, P
- The size (number of units) of the output volume will then be:

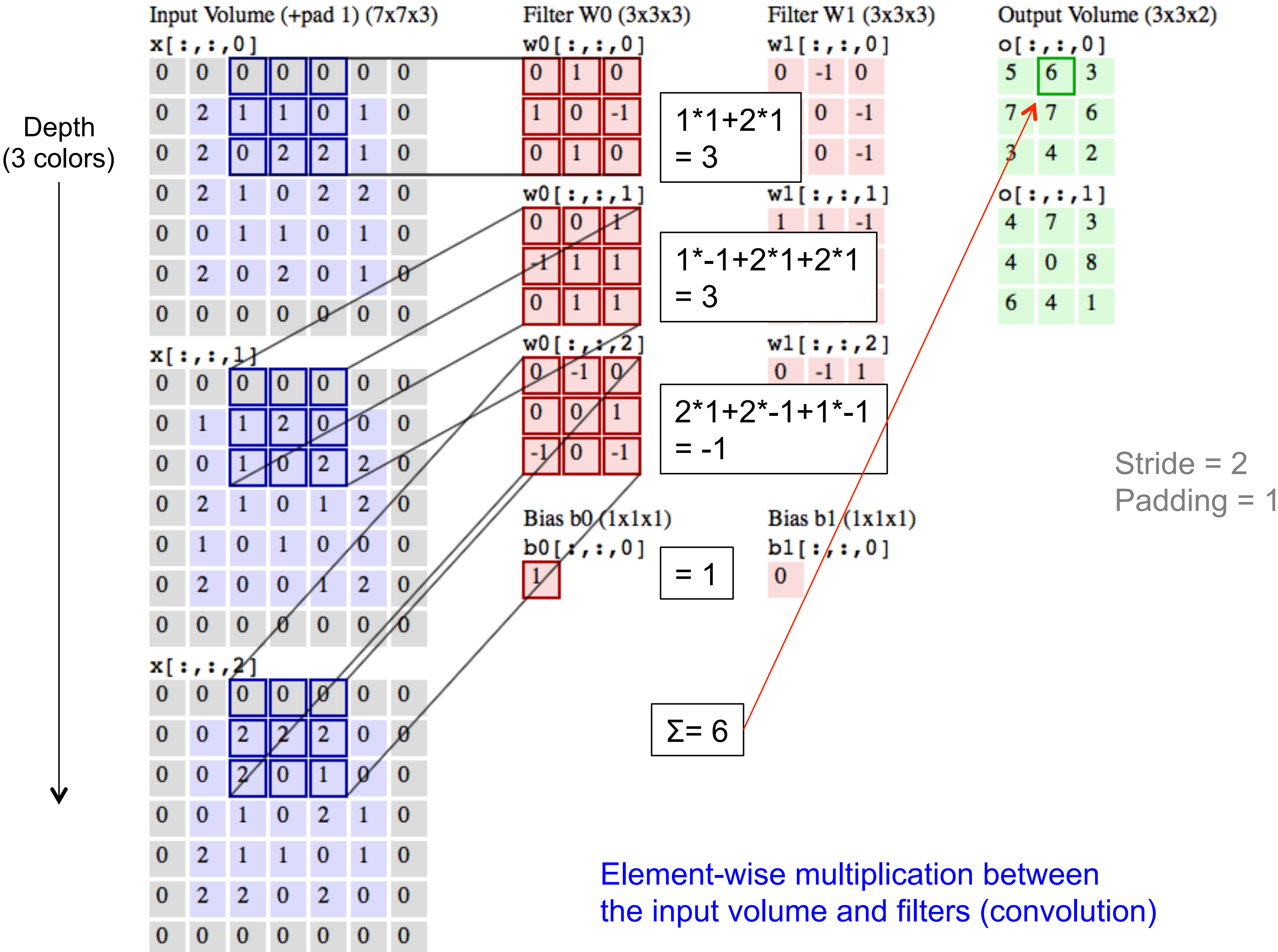
$$size = \frac{W - F + 2P}{S} + 1$$

Output volume

- Example:
  - Input volume is 32x32
  - Filters are 5x5
  - Stride is 1 and padding 0
  - Output volume is then 28x28 pixels (and depth depends on the number of filters we use)

Convolution

- Each depth slice uses the same weights (the weights of the filter) regardless of position in the input volume
- The forward pass can then be computed as a *convolution* of the unit's weights with the input volume
- That's why the layer is called a Conv layer



Element-wise multiplication between
the input volume and filters (convolution)

Filter examples

- Examples of filters learned by Krizhevsky et al. in the ImageNet challenge
- Each filter is 11x11 pixels and 3 color channels
- A total of 96 filters is used

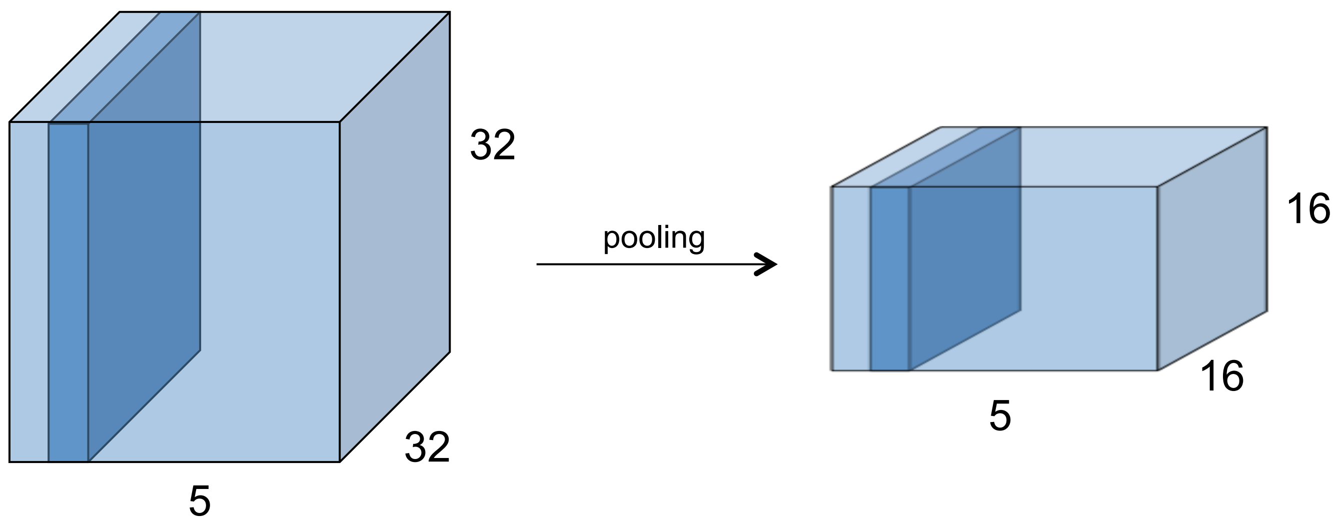
Pooling Layer

- Pooling layers are inserted between Conv layers
- The purpose is to reduce the size of the volumes, which reduces the number of weights needed and also controls overfitting
- The pooling layer acts independently on every depth slice of the input volume
- The width and height of each slice is reduced using the max operation

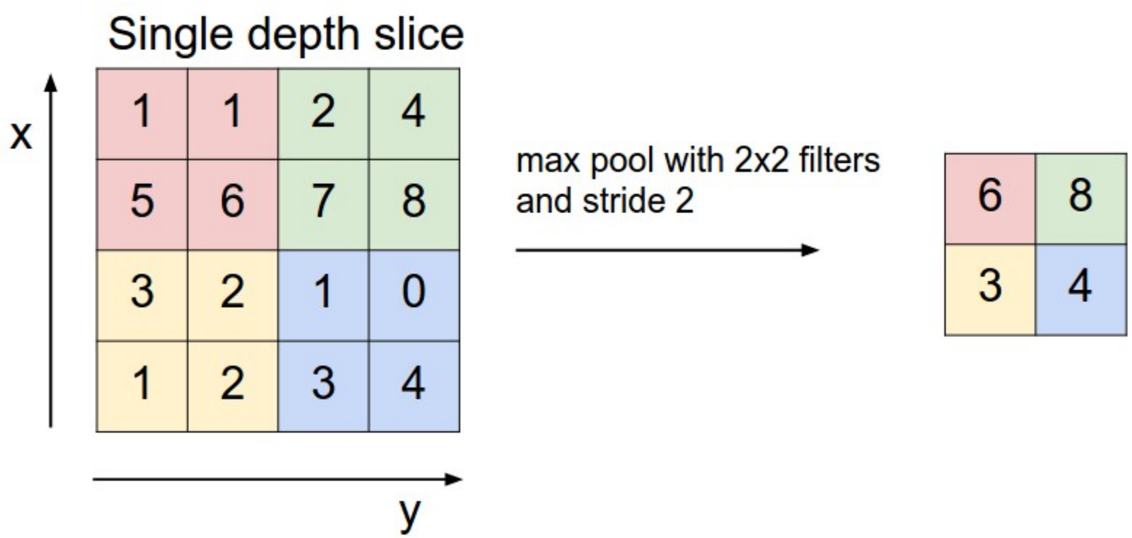
Pooling Layer

- The most common type of pooling layer is to use 2x2 filters with a stride of 2
- This cuts the width and height in half, and reduces activations with 75%
- The max operation takes the max value of 2x2 = 4 pixels

Pooling Layer



Pooling Layer



Fully-connected Layer

- A fully-connected layer works as the hidden layers in a regular NN
- The activation is a matrix multiplication followed by a bias offset

ReLU Layer

- We usually also write ReLU non-linearity as a layer
- It takes each value in the input volume, and calculates ReLU activation of that value:

$$f(x) = max(0, x)$$

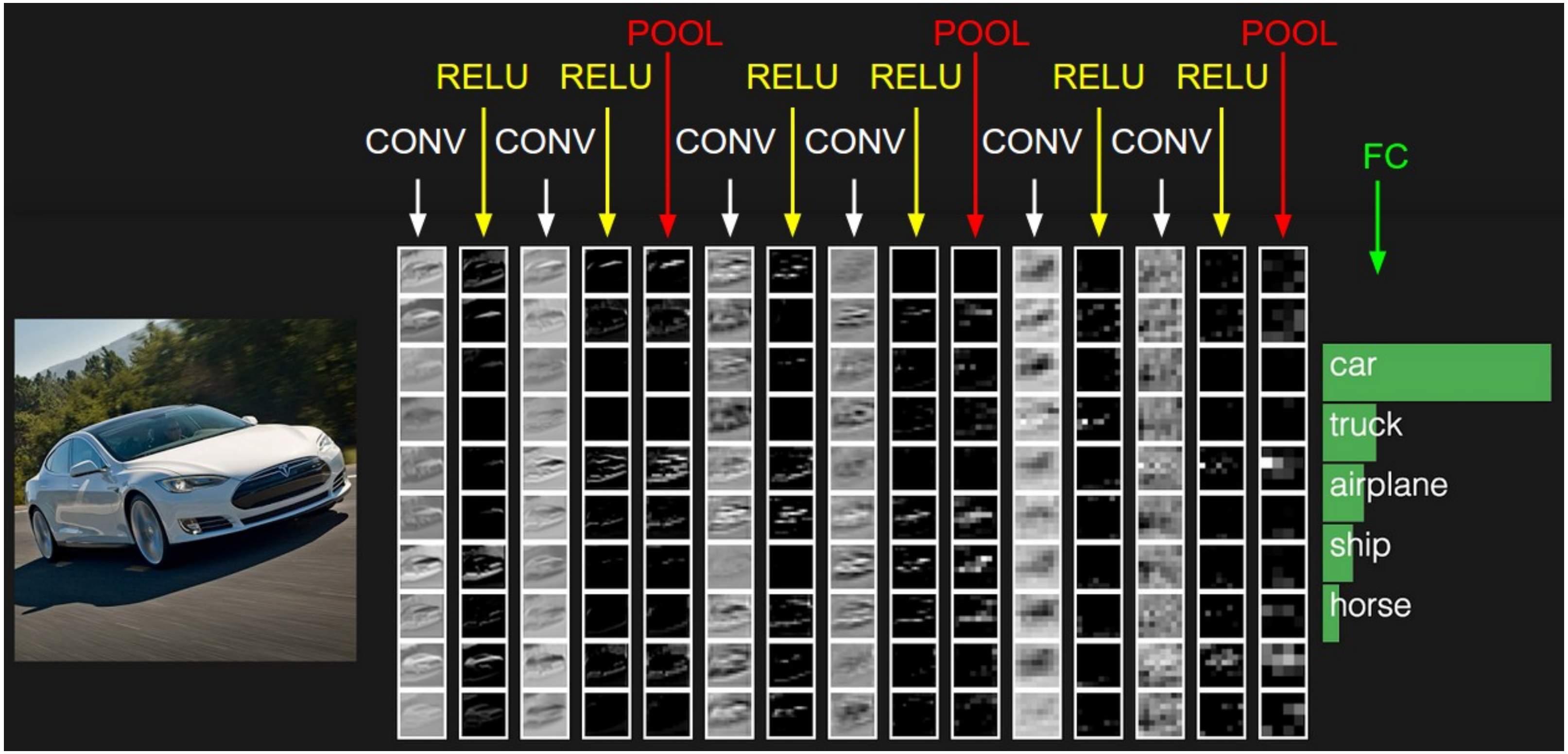
- No matrix operations are done in the ReLU layer

ConvNet Architectures

ConvNet Architectures

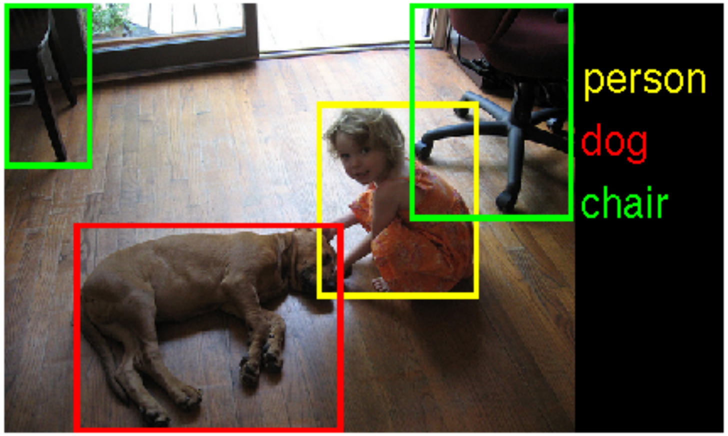
- A ConvNet is made up of:
  - Conv layers (CONV)
  - Pooling layers (POOL)
  - Fully-connected layers (FC)
  - ReLU non-linearity (RELU)
- The most common ConvNet architecture is:
  - Stacking a few CONV-RELU layers
  - Follow them with POOL layers
  - When the volume is of small enough size, transition to FC layers
  - The last layer is an output layer outputting a score for each category

Example Architecture



ImageNet challenge

- The ImageNet challenge is an annual contest for image classification and localization tasks
- The training dataset consists of 1.2 million images and 1000 possible categories
- The validation set for the challenge is a random subset of 50000 images
- Images can differ in size, but in average the resolution is 482x415 pixels
- ImageNet is the benchmark for image classification systems

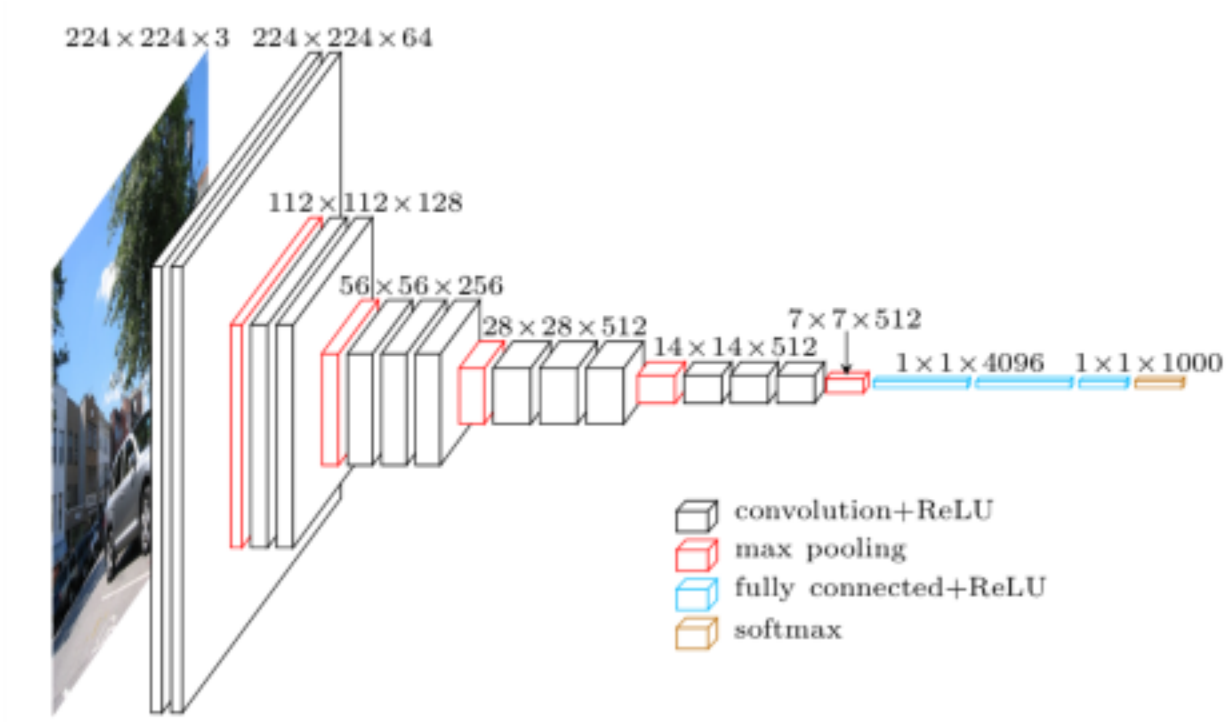


Standard Architectures

- There are several standardized architectures that have a name
- Some of them are:
  - LeNet: the first successful ConvNet developed int he 1990's
  - AlexNet: won the ImageNet challenge in 2012 by a wide margin
  - ZF Net: improvement of AlexNet that won the ImageNet challenge 2013
  - GoogLeNet: 2014 years winner
  - VGGNet: ended at second place in 2014 years ImageNet challenge
- Let's take a closer look at the VGGNet architecture:

Layer	Volume size	Description
INPUT	224x224x3	224x224 pixels and 3 color channels
CONV3-64 + ReLU	224x224x64	Conv layer with 64 3x3x3 filters
CONV3-64 + ReLU	224x224x64	Conv layer with 64 3x3x64 filters
POOL2	112x112x64	Standard 2x2 pooling layer with stride 2
CONV3-128 + ReLU	112x112x128	Conv layer with 128 3x3x64 filters
CONV3-128 + ReLU	112x112x128	Conv layer with 128 3x3x128 filters
POOL2	56x56x128	Standard 2x2 pooling layer with stride 2
CONV3-256 + ReLU	56x56x256	Conv layer with 256 3x3x128 filters
CONV3-256 + ReLU	56x56x256	Conv layer with 256 3x3x256 filters
CONV3-256 + ReLU	56x56x256	Conv layer with 256 3x3x256 filters
POOL2	28x28x256	Standard 2x2 pooling layer with stride 2
CONV3-512 + ReLU	28x28x512	Conv layer with 512 3x3x256 filters
CONV3-512 + ReLU	28x28x512	Conv layer with 512 3x3x512 filters
CONV3-512 + ReLU	28x28x512	Conv layer with 512 3x3x512 filters
POOL2	14x14x512	Standard 2x2 pooling layer with stride 2
CONV3-512 + ReLU	14x14x512	Conv layer with 512 3x3x512 filters
CONV3-512 + ReLU	14x14x512	Conv layer with 512 3x3x512 filters
CONV3-512 + ReLU	14x14x512	Conv layer with 512 3x3x512 filters
POOL2	7x7x512	Standard 2x2 pooling layer with stride 2
FC + ReLU	4096	Fully-connected layer with 4096 units
FC + ReLU	4096	Fully-connected layer with 4096 units
FC Softmax	1000	Output layer with 1000 possible categories

VGGNet



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 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

VGGNet

- In total VGGNet needs around 93 MB of memory per image for the forward pass, and around twice that for the backward pass
- In total the architecture has 138M parameters (weights and biases)
- We need to use GPUs to efficiently train the architecture
- Memory can however be an issue on many GPUs and we might need to use more memory-efficient architectures

Performance

- ConvNets have high memory and computational requirements
- The most important hardware is a GPU that is supported by the ConvNet library we use
- TensorFlow supports many Nvidia graphics cards, but rarely (if any) cards from other brands

Example: MNIST

MNIST dataset

- Each image is 28x28 pixels and 1 color channel (gray-scale)
- Training set of 60000 images
- Test set of 10000 images
- 10 categories

ConvNet for MNIST

Layer	Volume size	Description
INPUT	28x28x1	28x28 pixels and 1 color channel
CONV5-32 + ReLU	28x28x32	Conv layer with 32 5x5x1 filters
POOL2	14x14x32	Standard 2x2 pooling layer with stride 2
CONV5-64 + ReLU	14x14x64	Conv layer with 64 5x5x32 filters
POOL2	7x7x64	Standard 2x2 pooling layer with stride 2
FC	1024	Fully-connected layer with 1024 units
FC	10	Output layer with 10 possible categories

ConvNet in TensorFlow

- The script for creating and running the ConvNet on the MNIST dataset in TensorFlow is available here:
  - https://www.tensorflow.org/get_started/mnist/pros
- Training iterates 20000 times
- Each iteration trains on a batch of 50 images

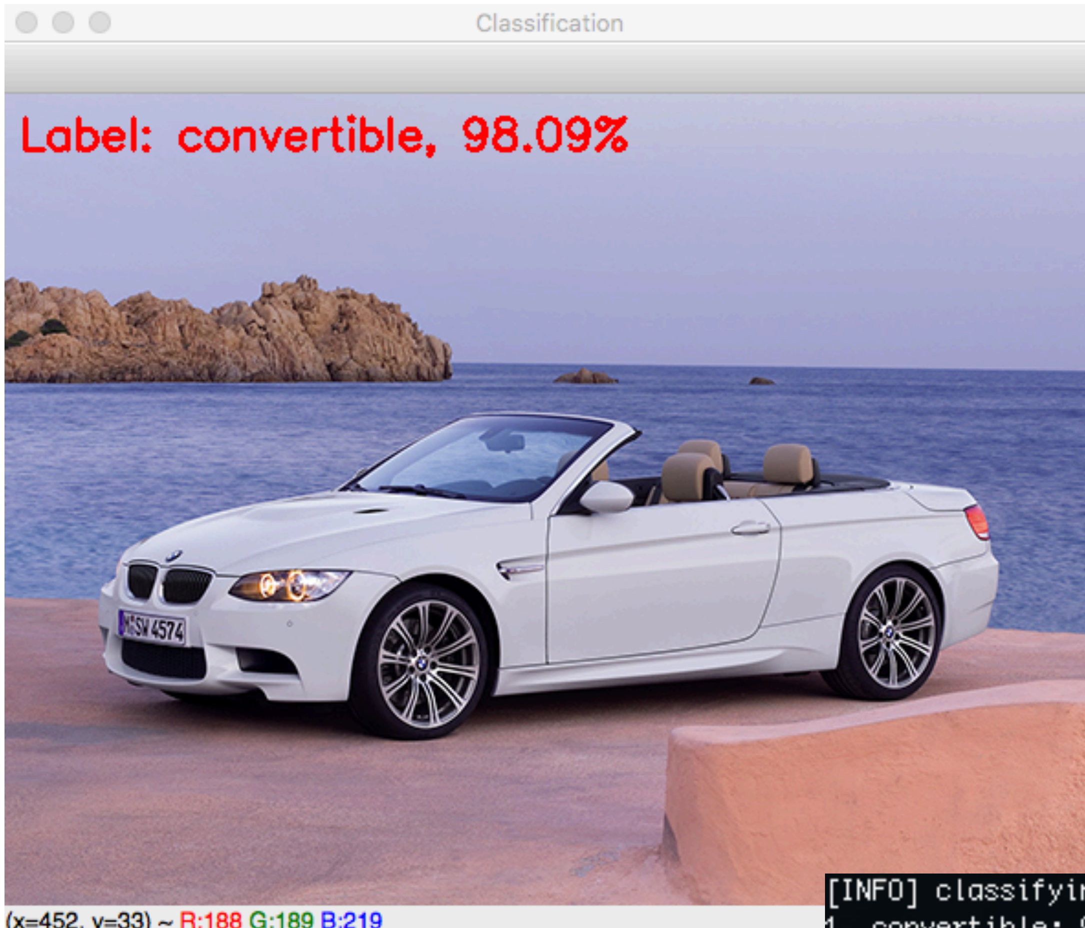
Results

- Training and evaluation took around 57 minutes on my Macbook Pro laptop
- The accuracy on the test set was 99.22%
- Compare this to a linear Softmax classifier
- Training and evaluation now took around 2 seconds and accuracy was 91.6%
- Using ConvNets on more complex image datasets requires expensive server hardware

Keras

- Keras is a high-level API running on top of DNN libraries, for example TensorFlow
  - https://keras.io/
- Keras is especially useful since it contains pre-trained ImageNet models, for example VGG16 and VGG19
- Training such models is extremely time consuming, so getting access to a pre-trained model can be very useful

Keras

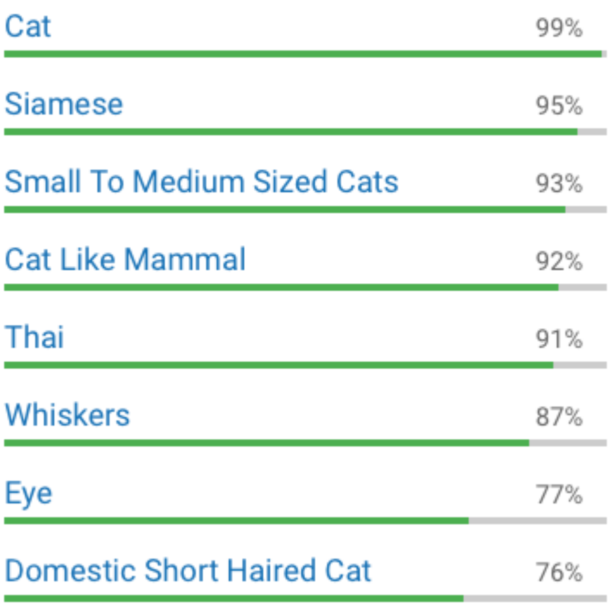



```
[INFO] classifying image with 'vgg16'...
1. convertible: 98.09%
2. sports_car: 0.63%
3. car_wheel: 0.43%
4. amphibian: 0.19%
5. beach_wagon: 0.18%
```

Google Vision API

Google Cloud Platform

https://cloud.google.com/vision/



Google Vision API

Google Cloud Platform

https://cloud.google.com/vision/

Google Vision API

Google Cloud Platform

https://cloud.google.com/vision/

Deep Learning

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org