# Visual and Acoustic Perception with Deep Neural Networks

M.Sc Matias Valdenegro-Toro

Ocean Systems Lab - Heriot-Watt University, Scotland (maybe UK)

June 28, 2016

# About Me

- PhD Student at Heriot-Watt University.
- Topic: Submerged Marine Debris Detection and Recognition with Deep Neural Networks.
- Research Interests: Robot and Underwater Perception, Deep Learning.

# Robocademy

ROBOCADEMY
European Academy for Marine and Underwater Robotics
ROBOCADEMY
EUROPEAN ACADEMY FOR MARINE AND UNDERWATER ROBOTICS
"This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no [FP7-PEOPLE-2013-ITN-608096]".

# Robocademy

- Three research lines: Autonomy, Perception and Disturbance Rejection.
- 13 Early Stage Researchers.
- Funded by a Marie-Curie Action (Initial Training Networks).
- Always open for collaborations!

# Introduction and Neural Network Basics

# Introduction

- Neural Networks are very old, have survived several periods of "no interest" (Neural Winters).
- Increasing interest lately due to large advances in some challenges, such as Image Classification and Speech Recognition.
- Enabled by the availability of large datasets, GPUs, and better models.

# Typical Machine Learning Pipeline


Handcrafted features can be: SIFT, SURF, HoG, LBP, or any kind of feature engineering. Trainable classifiers are typically SVMs, Random Forests, etc.

# Deep Learning Pipeline


In general, learning features from data beats feature engineering all the time.

# Why Learning Features is good?

- Domain adaptation.
- Exploits structures that might not be intuitive but still be present in data.
- Exploits the most relevant structures first.
- Relevant features are different for each problem.
- Practice moves faster than theory.

# What is Deep Learning?

- Hierarchical models that entirely learn features and classifiers from data.
- A feature hierarchy is learn from data.
- Depth is defined as number of stages that do non-linear feature extraction.
- Add more layers to a network!

# Feature Hierarchies

- Pixels $\rightarrow$ Edges $\rightarrow$ Textons $\rightarrow$ Parts $\rightarrow$ Objects.
- Character $\rightarrow$ Word $\rightarrow$ Word Groups $\rightarrow$ Sentence $\rightarrow$ Story.
- Sample $\rightarrow$ Spectral Band $\rightarrow$ Sound $\rightarrow$ Phoneme $\rightarrow$ Word.

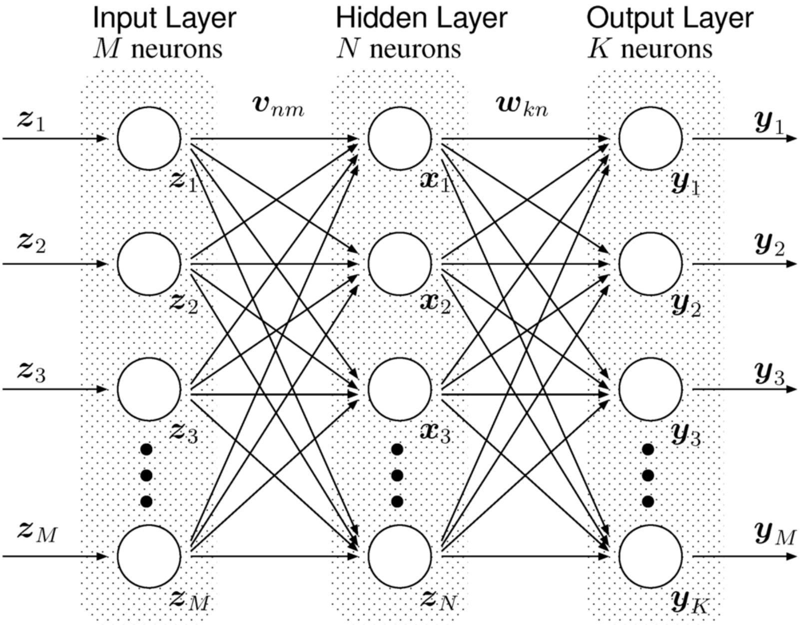
# Why Depth?

- Easier to train than "Wide" models.
- Requires less data.
- Learns the feature hierarchy. This cannot be done in Shallow or Wide models.
- Tradeoff between Parallel vs Sequential Computation.

# Neural Networks

- Extremely Non-Linear function approximation models.
- Universal Function Approximators.
- Adjustable Learning Capacity $\rightarrow O(N \log N)$ with $N$ total number of neurons in network.
- Considered "Deep" if more than two hidden layers.

# Multilayer Perceptron



# Multilayer Perceptron

$$z(\mathbf{x}) = \phi\left(\sum_i w_i x_i + b\right) = \phi(\mathbf{w} \cdot \mathbf{x} + b)$$

## Notation

- $z$: Activation value for input $\mathbf{x}$
- $\mathbf{w}$: Weight vector
- $b$: Bias value.
- $\phi(x)$: Activation function.

# Activation Functions

Any non-linear function can be used, its purpose is to introduce non-linearities into the network outputs. Ideally it must be differentiable.

## Sigmoid

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

## Hyperbolic Tangent

$$\phi(x) = \frac{e^{2x} - 1}{e^{-2x} + 1}$$

# Softmax Activation

Vector Activation function that takes a vector in any range and transforms it into a discrete probability distribution.

$$\mathrm{softmax}(\mathbf{x}) = \left\{ \frac{e^{x_j}}{\sum_i e^{x_i}} \right\}_j$$

For example, $\mathrm{softmax}(10, 3, 1) = [0.99, 0.0009, 0.0]$.
To output a specific class, take the one with biggest probability.

# Loss Functions

Assuming we have $n$ inputs $\mathbf{x}$ and same amount of target values $\mathbf{y}$ we can define a loss (error) function. Minimizing the loss function is equivalent to learning.

## Regression - Mean Square Error (MSE)

$$L(f(x), y) = \frac{1}{n} \sum_i (f(x_i) - y_i)^2$$

## Classification - Categorical Cross-Entropy

$$L(f(x), y) = -\sum_i \sum_c y_i^c \log(f^c(x_i))$$

# Minimizing the Loss Function

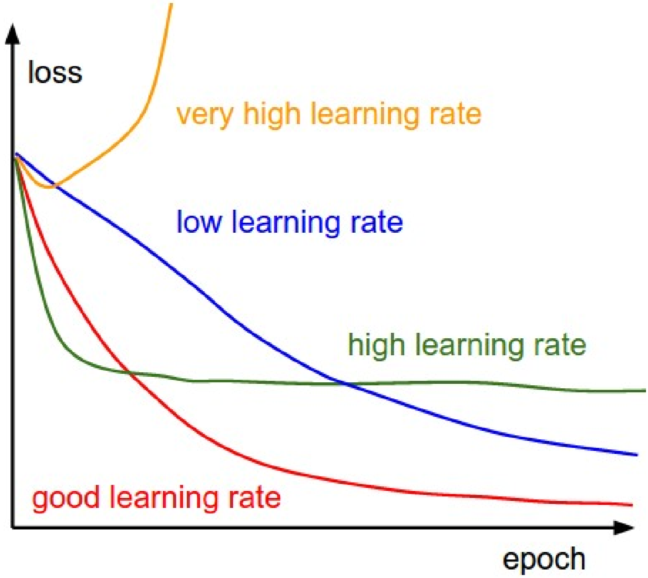
Minimization is with gradient descent. Weights are updated with the following rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla L(f(\mathbf{x}), \mathbf{y})$$

Weights are initially set to random values in the $[-1, 1]$ range. $\alpha \in [0, 1]$ is called the learning rate. The operator $\nabla$ computes the gradient of the Loss function with respect to the weights of the network.

$$\nabla L(f(\mathbf{x}, \mathbf{y})) = \left\{ \frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_m} \right\}$$

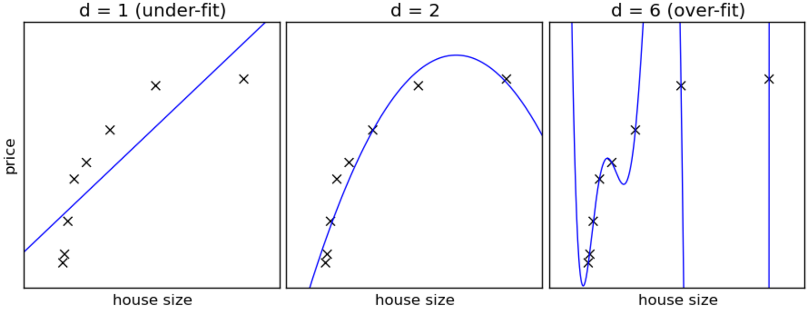
# Learning Rate



# Overfitting

- Overfitting is when the trained model memorizes undesirable patterns from training data.
- Like models learning noise in the data or irrelevant features.
- It reduces generalization performance. Models perform badly on the test set or on different data.
- Happens when model has too much learning capacity or it is trained for too long.

# Overfitting

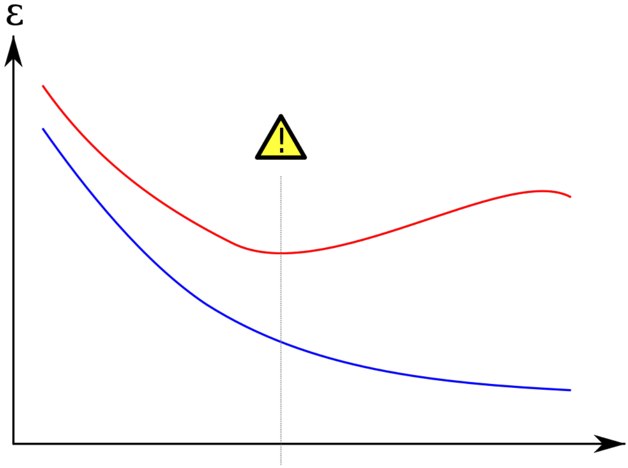


# Overfitting

## Golden Rule

When training a model, compare the loss value on the training set and on the testing set.
If the loss on the test set is **much larger** than in the training set, then the model is overfitting, specially if the training loss is low.
It is also normal that the test loss is **slightly** larger than training loss.

# Overfitting




Blue is training loss, Red is validation/test loss.

# How to Prevent Overfitting?

- Use regularization or methods that combat overfitting (Dropout and Batch Normalization).
- Train with more data.
- Reduce model learning capacity.
- Use early stopping. Stop training if **validation** loss is not improving.

# Regularization

It is a way to "guide" or introduce additional information to the learning process in order to reduce overfitting. The most common way is to introduce a new term to the loss function:

$$L^*(f(x), y) = L(f(x), y) + \lambda \sum_i |w_i|^p$$

$\lambda$ is a hyperparameter that defines the strength of regularization and can be computed with cross-validation. $p$ is typically 1 or 2.

# Universal Approximators

The Universal Approximation Theorem [1] states that MLPs with one hidden layer can learn any function to arbitrary precision. But it does not say:

- What network configuration can achieve it.
- How it can be trained.
- How much data is required.

[1] Cybenko., G. (1989) "Approximations by superpositions of sigmoidal functions"

# Issues with Multilayer Perceptrons

**Training Data Size** Approximately 20 times the number of neurons is required as data points for training.

**Overfitting** Large MLPs can represent very complex functions, and they are prone to overfit.

**Feature Engineering** MLPs do not easily extract relevant features from the data.

**Vanishing Gradient** Increasing the number of hidden layers results in diminishing gradients which makes the network hard to train.

**Black Box** It is not possible to easily interpret what the network has learned.

# Using MLPs with Image Inputs

- A $p$-neuron layer connected to an $n \times m$ image will have $p \times n \times m$ weights.
- Too many weights to be learned.
- The network completely ignores spatial correlation inside images.
- No translation invariance is built into the network design.
- In summary, this does not work.

# Convolutional Neural Networks

In the 80's Yann LeCun had the following idea:

- Connect a neuron only to a spatial neighborhood of the image and slide it on the image.
- This learns the same weights independent of location in the image. **Less weights to learn**.
- This is naturally represented as convolution, where the convolution filter contains the neuron weights.
- To reduce the amount of information, subsample the outputs after convolution.

# Example

For a $500 \times 500$ input image:

## One Perceptron Layer

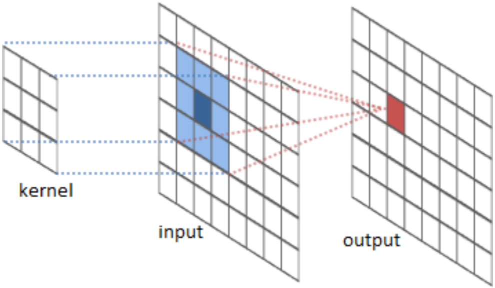
100 neurons. $500 \times 500 \times 100 = 25\text{M}$ parameters.

## Convolutional Layer

100 $5 \times 5$ filters. $100 \times 5 \times 5 = 2500$ parameters.

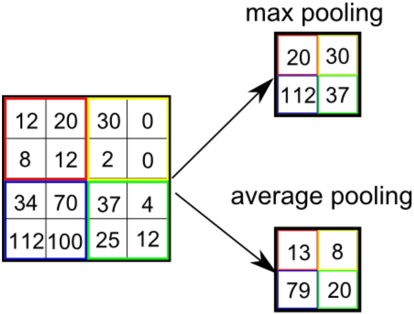
10K times less parameters to be learned. **Better**.

# Convolution




$$\text{out}(x, y) = \sum_i \sum_j \text{input}(x + i, y + j) k(i, j)$$

# Sub-sampling (Pooling)




Max-Pooling

$$\max_{i \in R} i(r_x, r_y)$$

Average-Pooling

$$n^{-1} \sum_{i \in R} i(r_x, r_y)$$

# Advantages

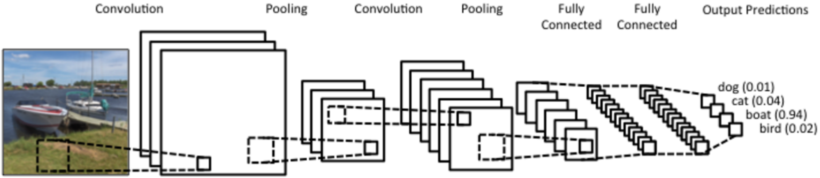
## Convolution

- Learned filters are translation invariant.
- Filters (kernels) are interpretable.
- Less parameters to be learned.

## Pooling

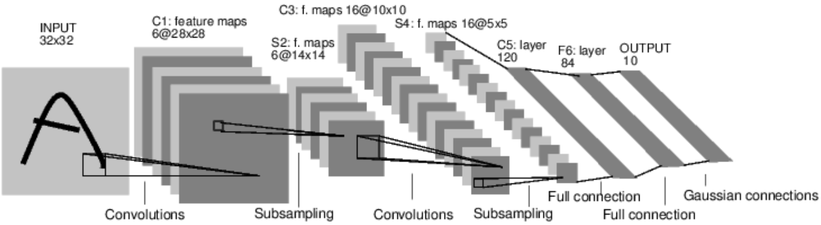
- Adds small degree of translation invariance.
- Reduces information but keeps important one.

# Convolutional Neural Network




- Multilayer Perceptron $\rightarrow$ Fully Connected Layer.
- Features are learned in Convolutional layers, Fully Connected layers act as classifiers.

# LeNet - 1989




Trained on the MNIST dataset for digit recognition, can achieve 99% accuracy.

# Network Configurations

To use a CNN to solve a task, one needs to define:

- Network architecture (layers, hyperparameters for each layer).
- Activation functions (specially at output layer).
- Loss functions.
- Train over a dataset, evaluate on a test set, and repeat until desired performance is achieved.

# Stochastic Gradient Descent

- Most dataset are large (10K, 100K). Evaluating and differentiating the loss is expensive.
- For really large datasets (millions) it is not feasible to load them into RAM.
- Solution is to run the network and compute the loss over a **batch** of samples.

$$L(f(x), y) = \frac{1}{b} \sum_{i}^{b} (f(x_i) - y_i)^2$$

$$L(f(x), y) = -\sum_{i}^{b} y_i \log(f(x_i))$$

# Stochastic Gradient Descent

- Introduces noise into the learning process but it is usually tolerable.
- For $b = 1$ it is called Stochastic Gradient Descent (SGD).
- For $b > 1$ it is called Mini-Batch Gradient Descent (MGD).
- Batch size $b$ has to be tuned as an extra hyperparameter.
- A whole pass over the a complete dataset is called an **epoch**.

# Data Augmentation

- CNN's require large amounts of data to be trained with good performance.
- New data can be generated by existing data, must be label invariant.
- Image rotations, flips (up down, left right), brightness changes.
- PCA, Scaling, Translating, etc.
- Choice completely depends on application, domain and available data.

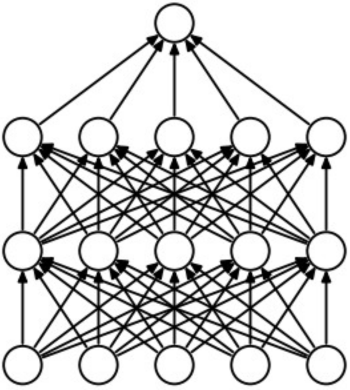
# Recent CNN Innovations

# Dropout

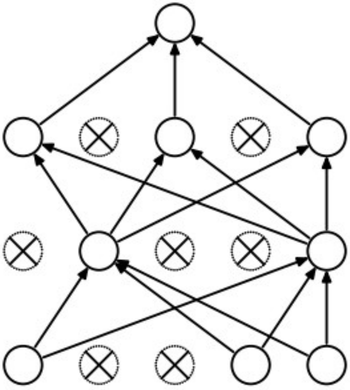
- Hinton et al. noticed that Neural Networks overfit due to "co-adaption" between neurons.
- One way to break co-adaption is to introduce noise into the network.
- Dropout layers can be introduced after Conv/FC layers.
- Each output from a layer is randomly set to zero with probability $p$.
- Common values are $p = 0.5$ and $p = 0.3$ [2].

[2] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." 2014

# Dropout



(a) Standard Neural Net



(b) After applying dropout.

# Batch Normalization

- Neural networks require normalized outputs, if not, training fails.
- Google researchers noticed that by normalizing inputs to layers, training is faster (less iterations) and it also introduces regularization.
- Batch Normalization layers compute:

$$x = \frac{x - E[x]}{\sqrt{\mathrm{Var}(x)}}$$

$$y = \gamma x + \beta$$

# Batch Normalization

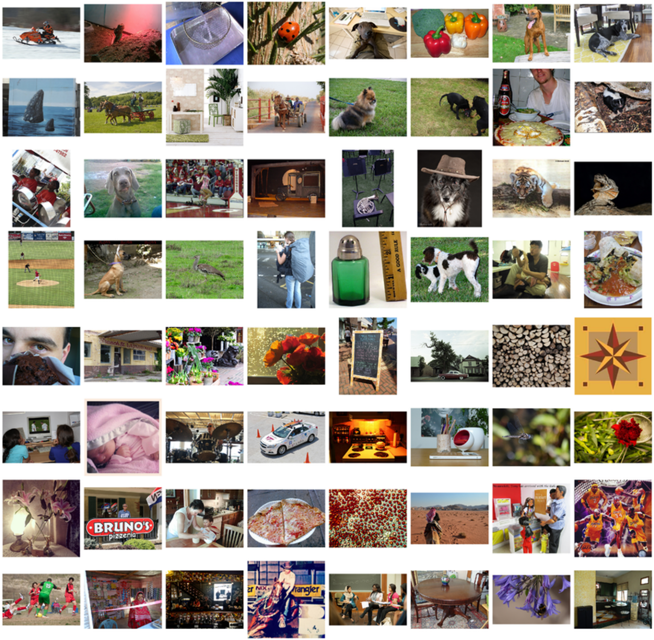
- Mean and Variance are computed at each batch. At test time population statistics are used.
- $\gamma$ and $\beta$ are learned scaling factors.
- Accelerates training by at least four times.
- Reduces the "co-variate shift" inside the network.
- Introduces regularization, improving performance [3].

[3] Ioffe, S., Szegedy, C. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." 2015

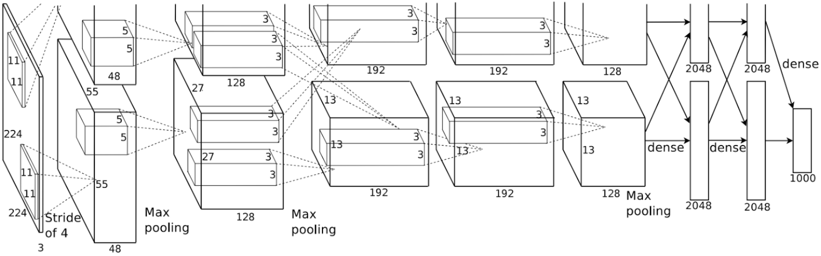
# ImageNet Dataset

- Dataset of 1.2 Million color images collected from the web.
- Labeled into 1000 different classes according to WordNet nouns.
- Defines different tasks on one challenge, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC):
  - Image Classification.
  - Object Detection.
  - Object Localization.

# ImageNet Dataset



# AlexNet - 2012



# AlexNet - 2012

- 5 Conv layers, 3 FC layers, 60 Million parameters. Softmax outputs.
- Trained on two GPUs for two weeks [4].
- Uses Dropout and ReLU activation function.

$$\mathrm{relu}(x) = \max(0, x)$$

- ILSVRC Top-5 error of 15.3%. Second place was 26.2%.

[4] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." 2012.

# AlexNet - 2012

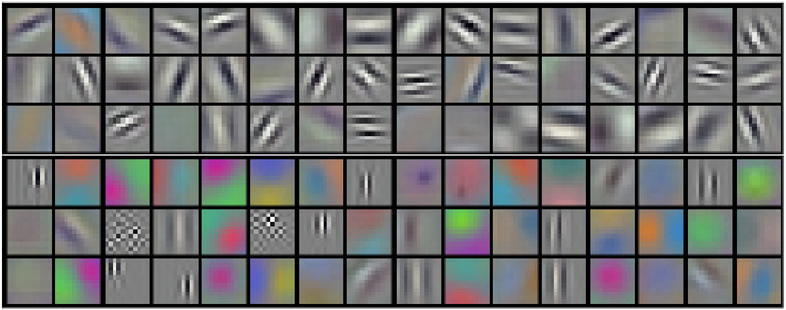
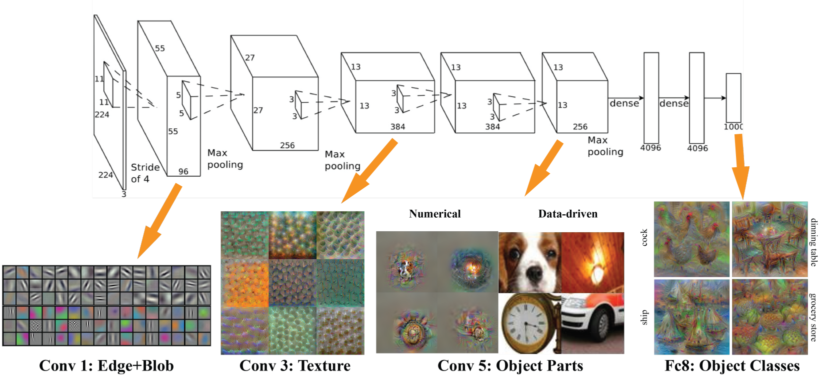
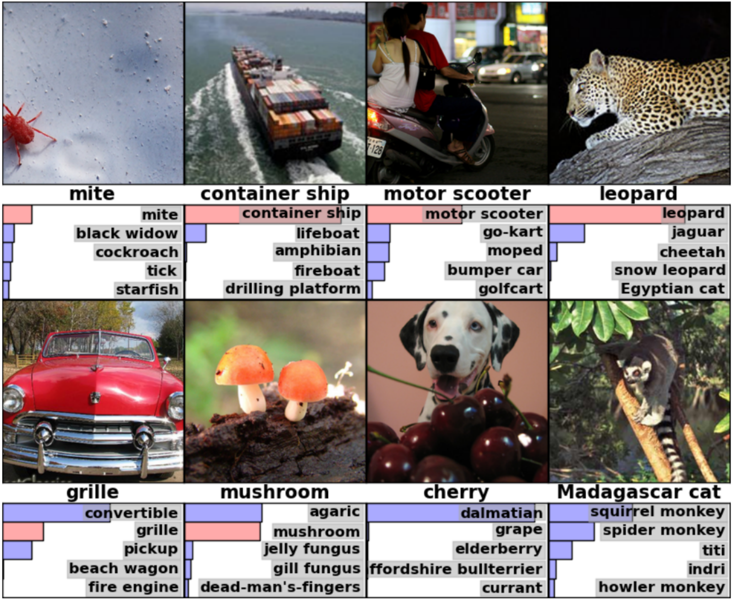


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The

# AlexNet - 2012



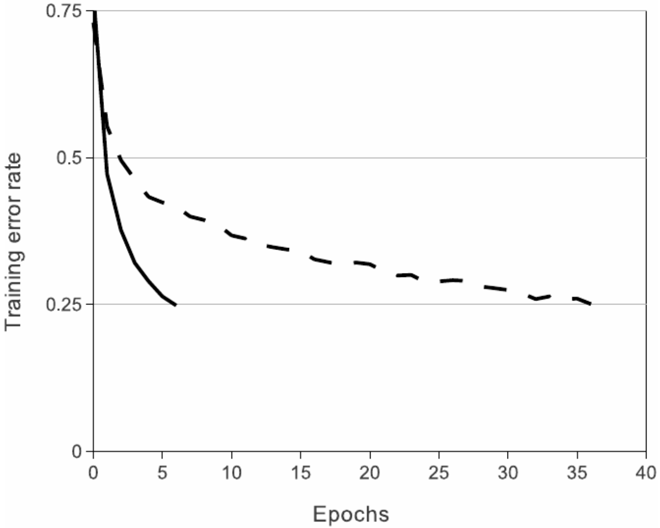
# AlexNet - 2012



# Why ReLU?

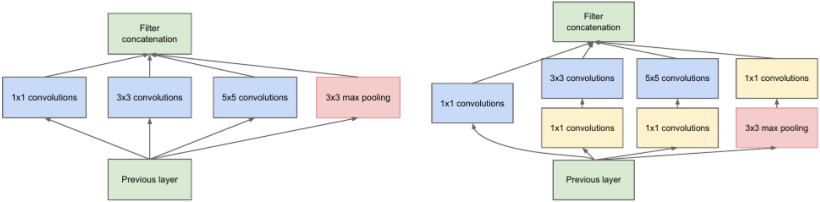
- ReLU = Rectifier Linear Unit.
- Sigmoid and Tanh activations saturate at their extremes.
- Saturation produces zero gradient $\rightarrow$ no learning.
- ReLU learns faster than other activations and does not saturate.

# Why ReLU?




Solid line is ReLU, Dashed line is Tanh.

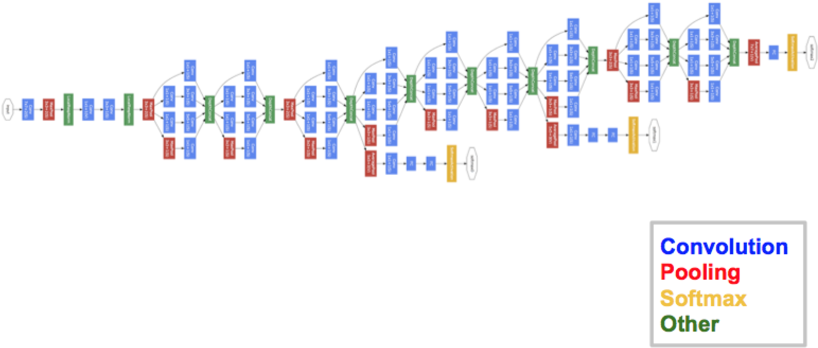
# GoogleNet - 2014




(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module

# GoogleNet - 2014



# GoogleNet - 2014

- 22 layers, 9 inception modules stacked.
- Inception modules represent more functions with less parameters and computation.
- 7 Network ensemble trained with 144 crops.
- 5 Million parameters [5].
- Top-5 error of 6.67%.

---

[5] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Rabinovich, A. "Going deeper with convolutions". 2015

# VGG - 2014

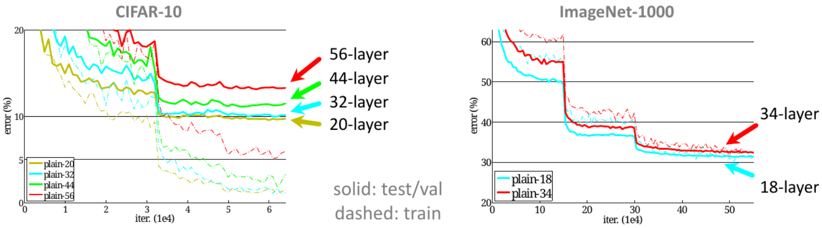
| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

# VGG - 2014

- Uses two $3 \times 3$ filters to emulate a $5 \times 5$ filter.
- 16-19 layers, only $3 \times 3$ filters are used.
- Configuration E has 144 Million parameters, Configuration A has 133 Million parameters.
- Top-5 error of 6.8% over a ensemble of two networks and multiple crops [6].

[6] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." 2014

# ResNet - 2015


CIFAR-10
error (%)
iter. (1e4)
plain-20
plain-32
plain-44
plain-56
56-layer
44-layer
32-layer
20-layer
solid: test/val
dashed: train
ImageNet-1000
error (%)
iter. (1e4)
plain-18
plain-34
34-layer
18-layer

# ResNet - 2015

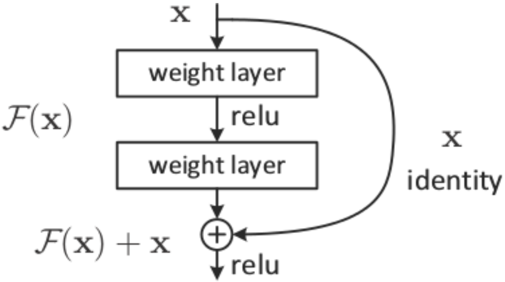
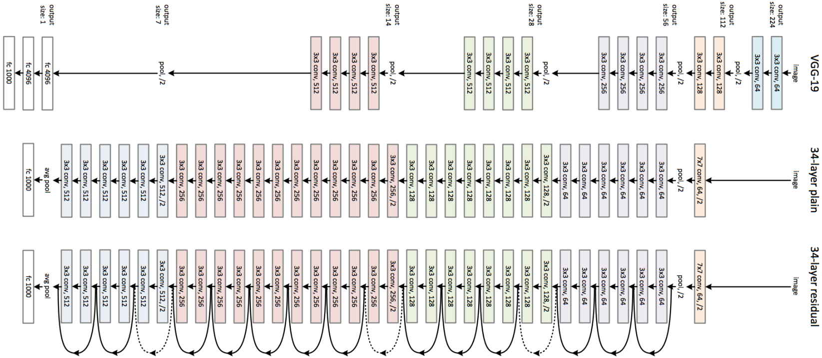



Figure 2. Residual learning: a building block.

# ResNet - 2015



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

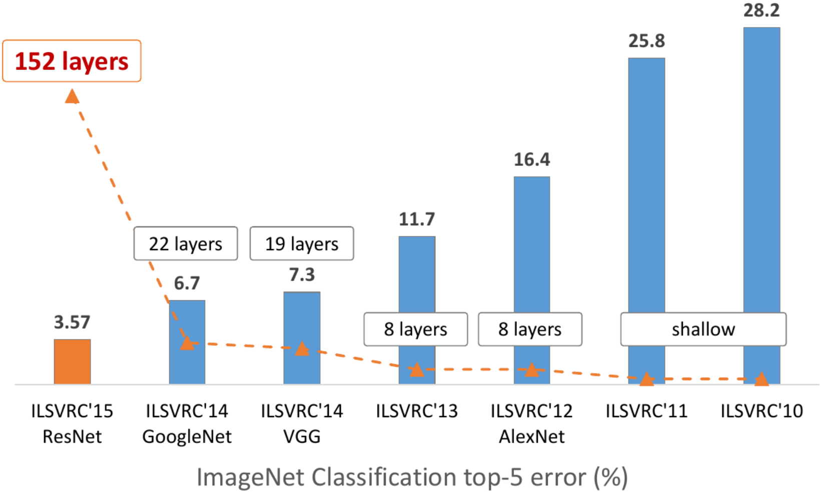
# ResNet - 2015

- ResNet for ILSVRC has 152 layers, approx 2.5 Million parameters.
- 3.57% Top-5 error.
- Authors tested a 1202-layer network for other purposes.
- The limit is now GPUs memory [7].

---

[7] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." 2015.

# ILSVRC Summary

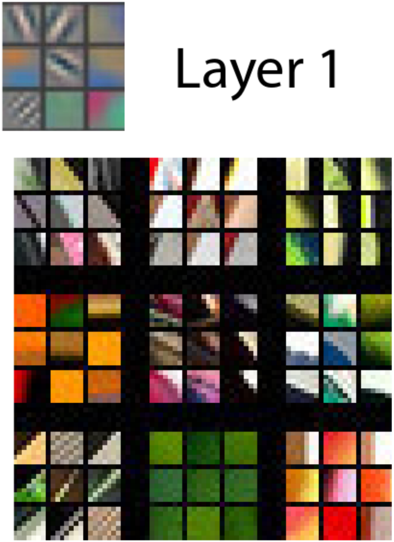


# Visualizing Deep Neural Networks

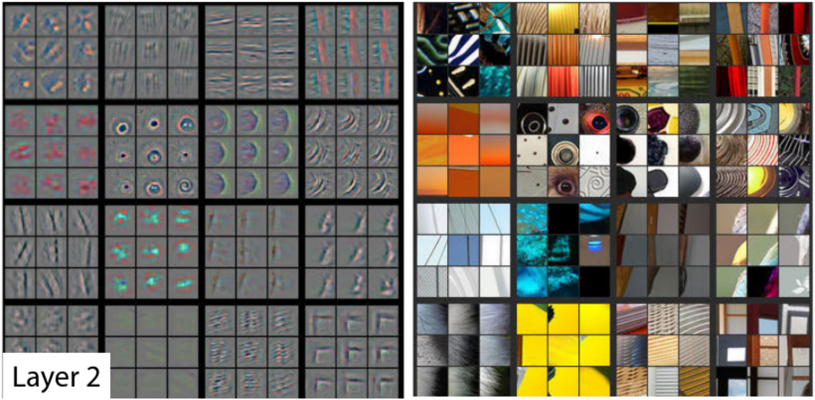
- There is always the question of validating or visualizing internal representation that a network learns.
- One way by activation maximization, select an output neuron and compute an image that maximizes the activation of that neuron.
- Deconvolution networks are also possible. [8]

[8] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks. 2014.

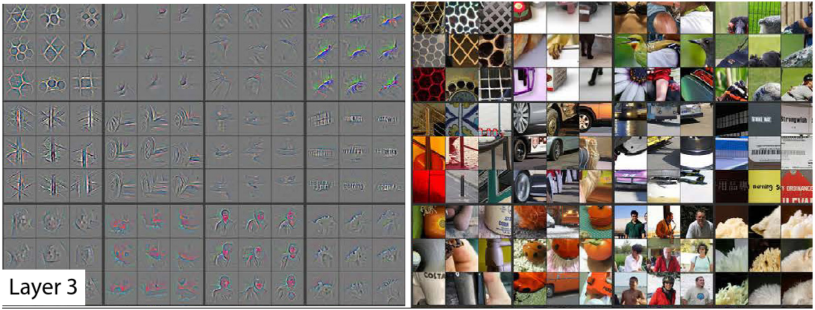
# Deconvolution Networks



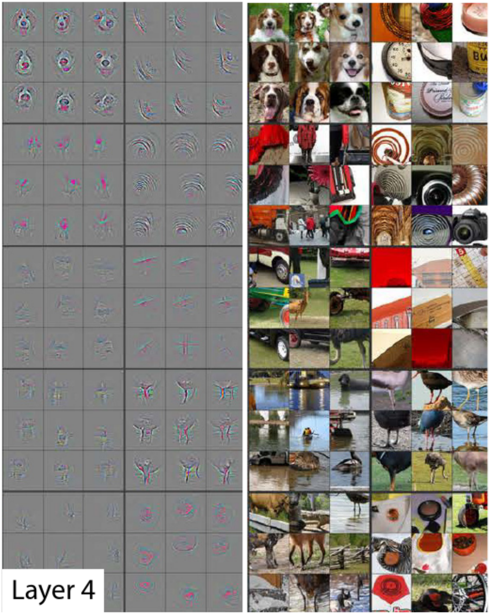
# Deconvolution Networks



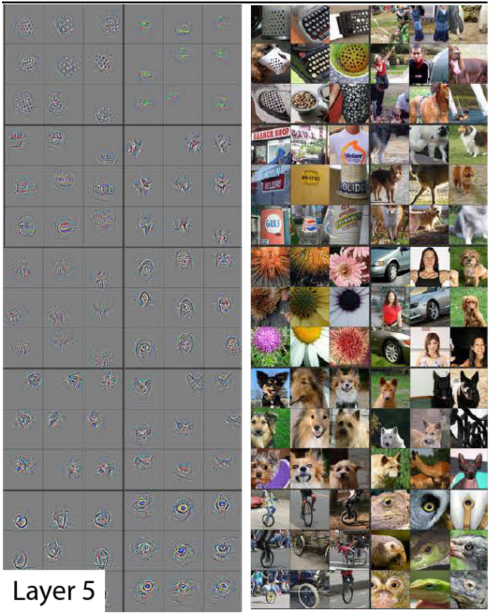
# Deconvolution Networks



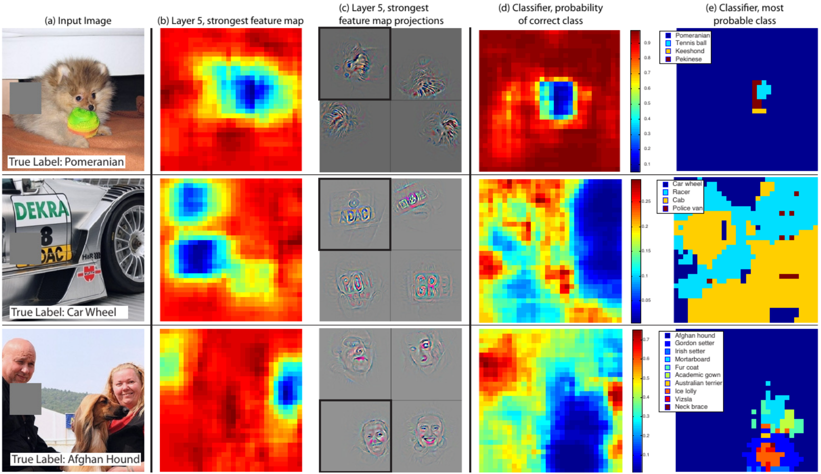
# Deconvolution Networks



# Deconvolution Networks



# Sensitivity Analysis



# Transfer Learning

- CNN's learn very good features that generalize well. Can these be transfered for a different task?
- Sharif et al [9] tested features extracted by a CNN and combined with a SVM classifier for different tasks.
- They take a 4096-dimensional feature vector from one of the fully connected layers.
- It most cases they get pretty close to state of the art, or beat it.

[9] Sharif Razavian, A., Azizpour, H., Sullivan, J. and Carlsson, S."CNN features off-the-shelf: an astounding baseline for recognition." 2014

# Transfer Learning

| | aero | bike | bird | boat | bottle | bus | car | cat | chair | cow | table | dog | horse | mbike | person | plant | sheep | sofa | train | tv | mAP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GHM[8] | 76.7 | 74.7 | 53.8 | 72.1 | 40.4 | 71.7 | 83.6 | 66.5 | 52.5 | 57.5 | 62.8 | 51.1 | 81.4 | 71.5 | 86.5 | 36.4 | 55.3 | 60.6 | 80.6 | 57.8 | 64.7 |
| AGS[11] | 82.2 | 83.0 | 58.4 | 76.1 | **56.4** | **77.5** | **88.8** | 69.1 | **62.2** | 61.8 | 64.2 | 51.3 | **85.4** | **80.2** | 91.1 | 48.1 | 61.7 | **67.7** | 86.3 | 70.9 | 71.1 |
| NUS[39] | 82.5 | 79.6 | 64.8 | 73.4 | 54.2 | 75.0 | 77.5 | 79.2 | 46.2 | 62.7 | 41.4 | 74.6 | 85.0 | 76.8 | 91.1 | 53.9 | 61.0 | 67.5 | 83.6 | 70.6 | 70.5 |
| CNN-SVM | 88.5 | 81.0 | 83.5 | 82.0 | 42.0 | 72.5 | 85.3 | 81.6 | 59.9 | 58.5 | 66.5 | 77.8 | 81.8 | 78.8 | 90.2 | 54.8 | 71.1 | 62.6 | 87.2 | 71.8 | 73.9 |
| CNNaug-SVM | **90.1** | **84.4** | **86.5** | **84.1** | 48.4 | 73.4 | 86.7 | **85.4** | 61.3 | **67.6** | **69.6** | **84.0** | **85.4** | 80.0 | **92.0** | **56.9** | **76.7** | 67.3 | **89.1** | **74.9** | **77.2** |

Table 1: **Pascal VOC 2007 Image Classification Results** compared to other methods which also use training data outside VOC. The CNN representation is not tuned for the Pascal VOC dataset. However, GHM [8] learns from VOC a joint representation of bag-of-visual-words and contextual information.

# Transfer Learning

| Method | mean Accuracy |
|---|---|
| ROI + Gist[36] | 26.1 |
| DPM[30] | 30.4 |
| Object Bank[24] | 37.6 |
| RBow[31] | 37.9 |
| BoP[21] | 46.1 |
| miSVM[25] | 46.4 |
| D-Parts[40] | 51.4 |
| IFV[21] | 60.8 |
| MLrep[9] | 64.0 |
| CNN-SVM | 58.4 |
| CNNaug-SVM | **69.0** |
| CNN(AlexConvNet)+multiscale pooling [16] | 68.9 |

Table 2: **MIT-67 indoor scenes dataset**. The MLrep [9] has a fine tuned pipeline which takes weeks to select and train various part detectors. Furthermore, Improved Fisher Vector (IFV) representation has dimensionality larger than 200K. [16] has very recently tuned a multi-scale orderless pooling of CNN features (off-the-shelf) suitable for certain tasks. With this simple modification they achieved significant average classification accuracy of **68.88**.

# Current Trends

- Combine Conv-ReLU-Max-Pooling. Stack these modules.
- Too much Max-Pooling can hurt performance by discarding too much information.
- Strided convolutions perform better.
- Batch Normalization > Dropout.
- Network architecture tuning is the new feature engineering.

# Neural Network Limitations

- Input sizes are usually fixed. There are some efforts to solve this.
- Computation is expensive, inference takes time.
- Power hungry (bad for underwater robotics).
- Large labeled datasets are required.
- Lots of hyperparmeters to decide and tune.

# Recommendations

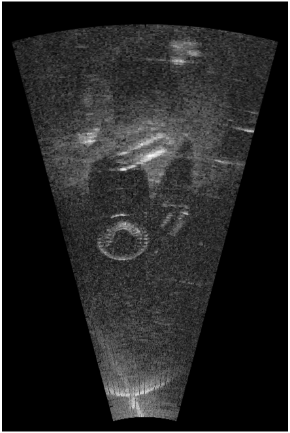
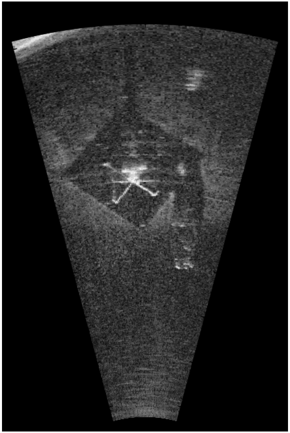
- Use grid or random search to decide some hyperparameters.
- Tune the right learning rate for your network/data.
- Use Batch Normalization and ADAM.
- Make a learning rate schedule, learning rate should be reduced as training advances.
- Use data augmentation to generate extra data.

# Applications to the Underwater Domain

# Motivation: Submerged Marine Debris

Lots of submerged garbage in coastal areas and deep sea. Garbage detection by AUVs requires advanced object detection and recognition capabilities.


# High Resolution Forward-Looking Sonar

# Convolutional Neural Networks on FLS


Conv(32, $5 \times 5$) $\rightarrow$ MP(2, 2) $\rightarrow$ Conv(32, $5 \times 5$) $\rightarrow$ MP(2, 2) $\rightarrow$ FC(64) $\rightarrow$ FC(classes)

# Sonar Image Classification

- Crop objects from FLS image and resize/scale to $64 \times 64$.
- Train CNN over this dataset, with 2500 objects and 10 different classes.
- Data augmentation: 15 Rotations, $\pm 2$ pixel offsets, 60 combinations total.
- Augmented data set is 150K images.

# Template Matching

- We select random examples as templates.
- The example with the biggest similarity is output as class.

## Cross-Correlation

$$S = (N_x N_y)^{-1} \sum_x \sum_i T_{xy} I_{xy}$$

## Sum of Squared Differences

$$S = (N_x N_y)^{-1} \sum_x \sum_i (T_{xy} - I_{xy})^2$$

# Sonar Image Classification

| Method | | | Accuracy | # of Parameters |
|---|---|---|---|---|
| Network type | Regularizer | Activation | | |
| CNN | BatchNorm | ReLU | 99.2 % | 930K |
| CNN | Dropout | ReLU | 96.9 % | 930K |
| MLP | Dropout | Sigmoid | 62.1 % | 10.1M |
| MLP | Dropout | Tanh | 16.5 % | 10.1M |
| MLP | Dropout | ReLU | 51.7 % | 10.1M |
| MLP | BatchNorm | Sigmoid | 65.5 % | 10.1M |
| MLP | BatchNorm | Tanh | 29.5 % | 10.1M |
| MLP | BatchNorm | ReLU | 92.3 % | 10.1M |
| TM with CC | | | 92.4% | 8.5M |
| TM with SQD | | | 97.6% | 8.5M |

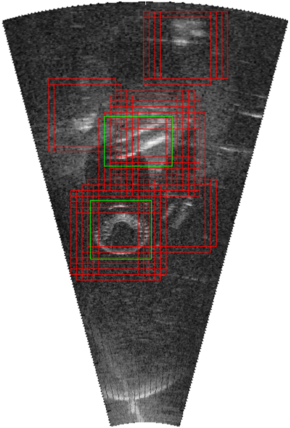
# Sonar Proposal Generation

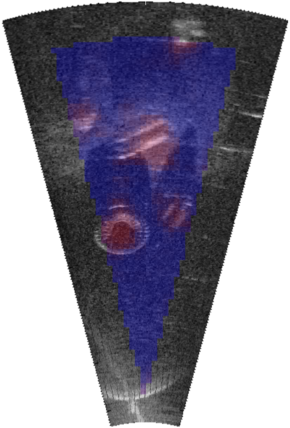
- Detection Proposals are generic object detectors on images.
- We trained a neural network that outputs an objectness score in $[0, 1]$. Ground truth of this score is computed from Intersection-over-Union (Iou) score:

$$\text{IoU}(A, B) = \frac{\text{area}(A \cap B)}{\text{area}(A \cup B)}$$

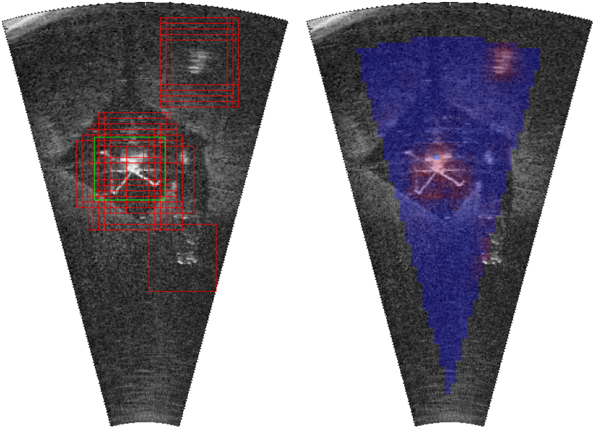
- $96 \times 96$ sliding window over the image.

# Sonar Proposals with a CNN

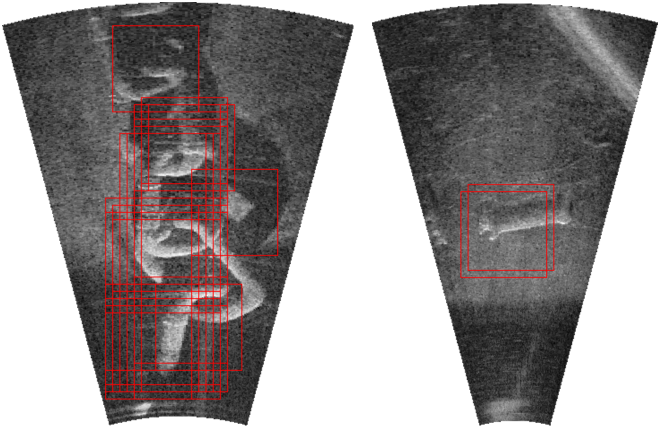




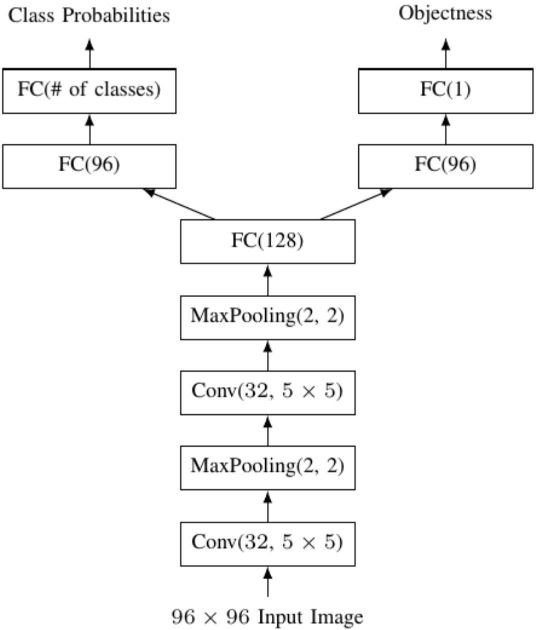
# Sonar Proposals with a CNN



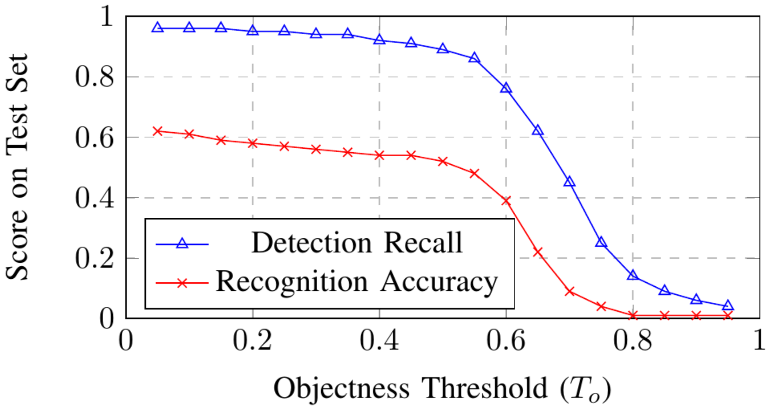
# Non-Trained Objects



# End-to-End Detection and Recognition



# End-to-End Detection and Recognition



# Neural Network Implementations

# Neural Network Software

**Symbolic Computation**

Theano, Tensorflow, Lasagne, Keras, etc.

**Non-Symbolic Computation**

Caffe, Matconvnet, DeepLearning4J, etc.

# Symbolic Computation

- Mostly on Python (and Tensorflow in C++).
- Automatic gradient computation through automatic differentiation.
- Very easy to use, easy to explore network internals and to experiment.
- Sometimes error messages are pretty confusing!
- Automatic use of the GPU if configured.

# Theano

```
import theano
import theano.tensor as T
x = T.dmatrix('x')
s = 1 / (1 + T.exp(-x))
logistic = theano.function([x], s)
logistic([[0, 1], [-1, -2]])

array([[ 0.5        ,  0.73105858],
[ 0.26894142,  0.11920292]])
```

# Keras

- https://github.com/fchollet/keras
- Pretty nice Deep Learning API. Lots of examples.
- Theano and Tensorflow Backends.
- Easy to use and useful for rapid prototyping.

# Keras

```
from keras.layers import Dense, Activation
from keras.models import Sequential

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100,
 activation = "relu"))
model.add(Dense(output_dim=10,
 activation = "softmax"))
model.compile(loss='categorical_crossentropy',
 optimizer='sgd', metrics=['accuracy'])
model.fit(X_train, Y_train, nb_epoch=5,
 batch_size=32)
```

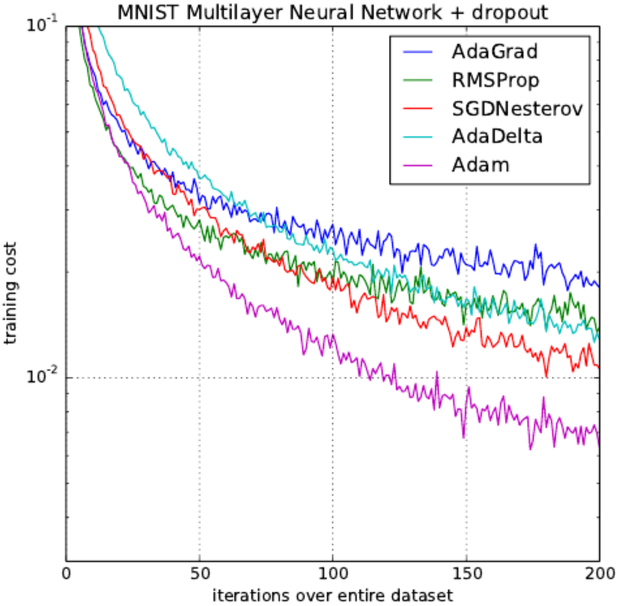
# Available Layers

- Dense, Dropout, Flatten, Reshape, Merge.
- Convolution1D, 2D, 3D, MaxPooling1D, 2D, 3D, AveragePooling1D, 2D, 3D.
- GRU, LSTM.
- BatchNormalization.
- GaussianNoise, GaussianDropout.

# Optimizers

- SGD (the default).
- RMSprop.
- Adagrad, Adadelta.
- ADAM (the best).

# Optimizer Comparison



# Functional API

- Keras' Sequential API is... well sequential.
- The functional API allows for graph-like connections between layers and nodes.
- Multiple inputs, and/or multiple outputs.
- Training is performed by a multi-task loss that is the weighted sum each output's loss.

# Functional API

```
input_img = Input(shape=(3, 256, 256))
tower_1 = Convolution2D(64, 1, 1)(input_img)
tower_1 = Convolution2D(64, 3, 3)(tower_1)

tower_2 = Convolution2D(64, 1, 1)(input_img)
tower_2 = Convolution2D(64, 5, 5)(tower_2)
tower_3 = MaxPooling2D((3, 3),
 strides=(1, 1))(input_img)
tower_3 = Convolution2D(64, 1, 1)(tower_3)

output = merge([tower_1, tower_2, tower_3],
 mode='concat', concat_axis=1)
```

# Functional API

```
# input tensor for a 3-channel 256x256 image
x = Input(shape=(3, 256, 256))
# 3x3 conv with 3 output channels
y = Convolution2D(3, 3, 3,
 border_mode='same')
# this returns x + y.
z = merge([x, y], mode='sum')
```

# The Future of Deep Learning

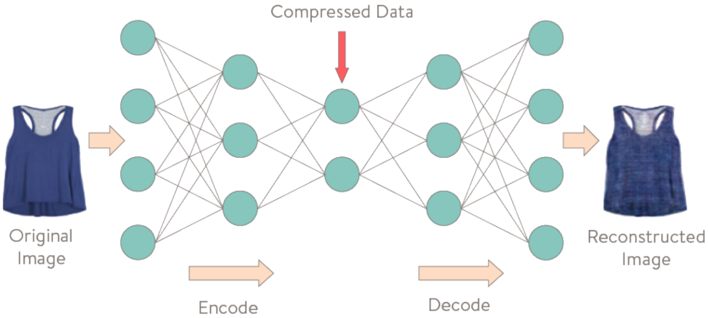
# Binary Neural Networks

- Replace floating point computation with binary operations.
- Threshold weights and convert to $\pm 1$ values.
- Convolution and FC layers can then be implemented with XOR bitops.
- Small loss of accuracy but computational performance improvement of 7-8x.

# Autoencoders

- Unsupervised method that learns to code a representation of the input.
- Neural network that predicts its input, but with a bottleneck hidden layer.
- Bottleneck layer forces the network to learn a useful representation of the input.
- Useful to discover structure in the data, without any labels.s

# Autoencoders



# Autoencoders

# Variational Autoencoders

- Usually one cannot influence the code that an autoencoder learns.
- A variational autoencoder can make such influence by imposing a distribution constraint on the code representation.
- This is done by adding a term to the loss function. The Kullback-Leibler Divergence between the coding distribution and a target distribution is added.

# Recurrent Neural Networks

- Neural network that can have connection loops and recurrence.
- They allow to have "memory" inside the network.
- Some applications: Sequence processing, text generation, image captioning.

# Open Research Questions

- Variable-sized inputs.
- Hyperparameter tuning (network structure, learning rate, regularization, etc).
- Theory lags behind practice.
- Why do Deep Neural Networks work?

# Recommended Sites

- http://cs231n.github.io/
- http://www.keras.io
- https://github.com/Theano/Theano
- http://www.tensorflow.org

# Recommended Literature

Questions?