Sistemas Inteligentes

Convolutional Neural Networks

José Eduardo Ochoa Luna
Dr. Ciencias - Universidade de São Paulo

Maestría C.C. Universidad Católica San Pablo
Sistemas Inteligentes

14 de diciembre 2017

Deep Learning Introduction

- Allow computers to learn from experience and understand the world in terms of a hierarchy of concepts

Deep Learning Introduction

- Allow computers to learn from experience and understand the world in terms of a hierarchy of concepts
- Each concept defined in terms of its relations to simpler concepts

Deep Learning Introduction

- Allow computers to learn from experience and understand the world in terms of a hierarchy of concepts
- Each concept defined in terms of its relations to simpler concepts
- The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones

Deep Learning Introduction

- Allow computers to learn from experience and understand the world in terms of a hierarchy of concepts
- Each concept defined in terms of its relations to simpler concepts
- The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones
- If we draw a graph showing how these concepts are built on top of each other, the graph is deep with many layers: AI Deep Learning

Representation

- Difficulties faced by systems relying on hard-coded knowledge (logic) suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data : Machine Learning

Representation

- Difficulties faced by systems relying on hard-coded knowledge (logic) suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data : Machine Learning
- The performance of machine learning algorithms depends heavily on the representation of the data they are given

Representation

- Difficulties faced by systems relying on hard-coded knowledge (logic) suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data : Machine Learning
- The performance of machine learning algorithms depends heavily on the representation of the data they are given
- The choice of representation has an enormous effect on the performance of machine learning algorithms

Features

- Many AI tasks can be solved by designing the right set of features

Features

- Many AI tasks can be solved by designing the right set of features
- A useful feature for speaker identification from sound is an estimate of the size of speaker's vocal tract

Features

- Many AI tasks can be solved by designing the right set of features
- A useful feature for speaker identification from sound is an estimate of the size of speaker's vocal tract
- It gives a strong clue as to whether the speaker is a man, woman, or child

Features

- Many AI tasks can be solved by designing the right set of features
- A useful feature for speaker identification from sound is an estimate of the size of speaker's vocal tract
- It gives a strong clue as to whether the speaker is a man, woman, or child
- For many tasks, it is difficult to know what features should be extracted: detect cars in photographs

Learning Representation

- One solution to this problem is to use ML to discover not only the mapping from representation to output but also the representation itself

Learning Representation

- One solution to this problem is to use ML to discover not only the mapping from representation to output but also the representation itself
- This approach is known as learning representation

Learning Representation

- One solution to this problem is to use ML to discover not only the mapping from representation to output but also the representation itself
- This approach is known as learning representation
- Representations Learned often result in much better performance than can be obtained with hand-designed representations

Learning Representation

- One solution to this problem is to use ML to discover not only the mapping from representation to output but also the representation itself
- This approach is known as learning representation
- Representations Learned often result in much better performance than can be obtained with hand-designed representations
- Manually designing features for a complex task requires a great deal of human time and effort; it can take decades for an entire community of researchers

Deep Learning

Deep-learning methods are learning representation methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level.

Deep Learning Architecture

- A deep-learning architecture is a multilayer stack of simple modules, all (or most) of which are subject to learning, and many of which compute non-linear input/output mappings.

Deep Learning Architecture

- A deep-learning architecture is a multilayer stack of simple modules, all (or most) of which are subject to learning, and many of which compute non-linear input/output mappings.
- Each module in the stack transforms its input to increase both the selectivity and the invariance of the representation.

Deep Learning Architecture

- A deep-learning architecture is a multilayer stack of simple modules, all (or most) of which are subject to learning, and many of which compute non-linear input/output mappings.
- Each module in the stack transforms its input to increase both the selectivity and the invariance of the representation.
- With multiple non-linear layers, say a depth of 5 to 20, a system can implement extremely intricate functions of its inputs that are simultaneously sensitive to minute details and

Deep Learning Architecture

- A deep-learning architecture is a multilayer stack of simple modules, all (or most) of which are subject to learning, and many of which compute non-linear input/output mappings.
- Each module in the stack transforms its input to increase both the selectivity and the invariance of the representation.
- With multiple non-linear layers, say a depth of 5 to 20, a system can implement extremely intricate functions of its inputs that are simultaneously sensitive to minute details and
- Insensitive to large irrelevant variations such as the background, pose, lighting and surrounding objects.

Deep Learning Applications

- Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years.

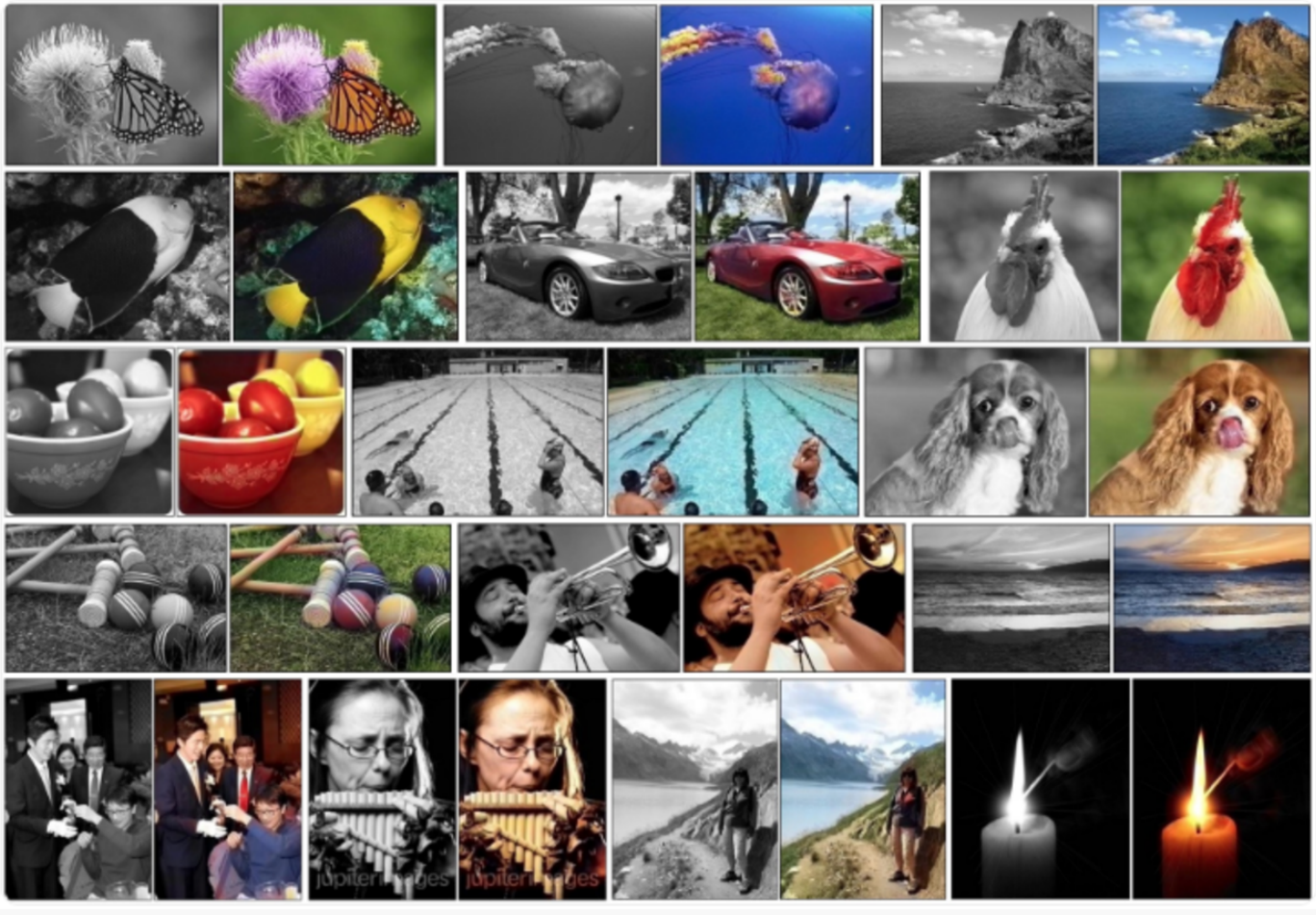
Deep Learning Applications

- Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years.
- It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government.

Deep Learning Applications II

Deep learning has produced extremely promising results for various tasks in natural language understanding, particularly topic classification, sentiment analysis, question answering and language translation

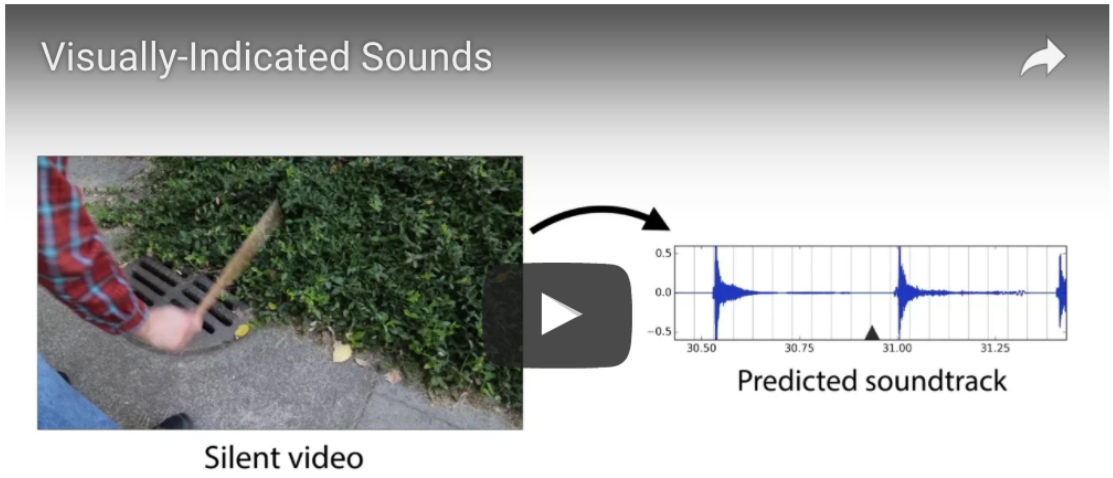
Applications: Colorization of Black and White Images



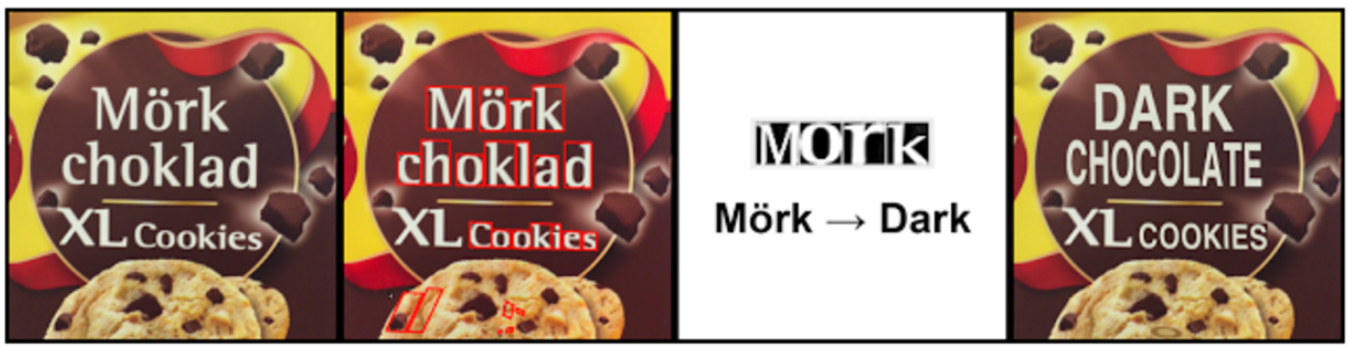
Colorization of Black and White Photographs

Image taken from Richard Zhang, Phillip Isola and Alexei A. Efros.

Applications: Adding Sounds To Silent Movies



Applications: Automatic Machine Translation



Instant Visual Translation

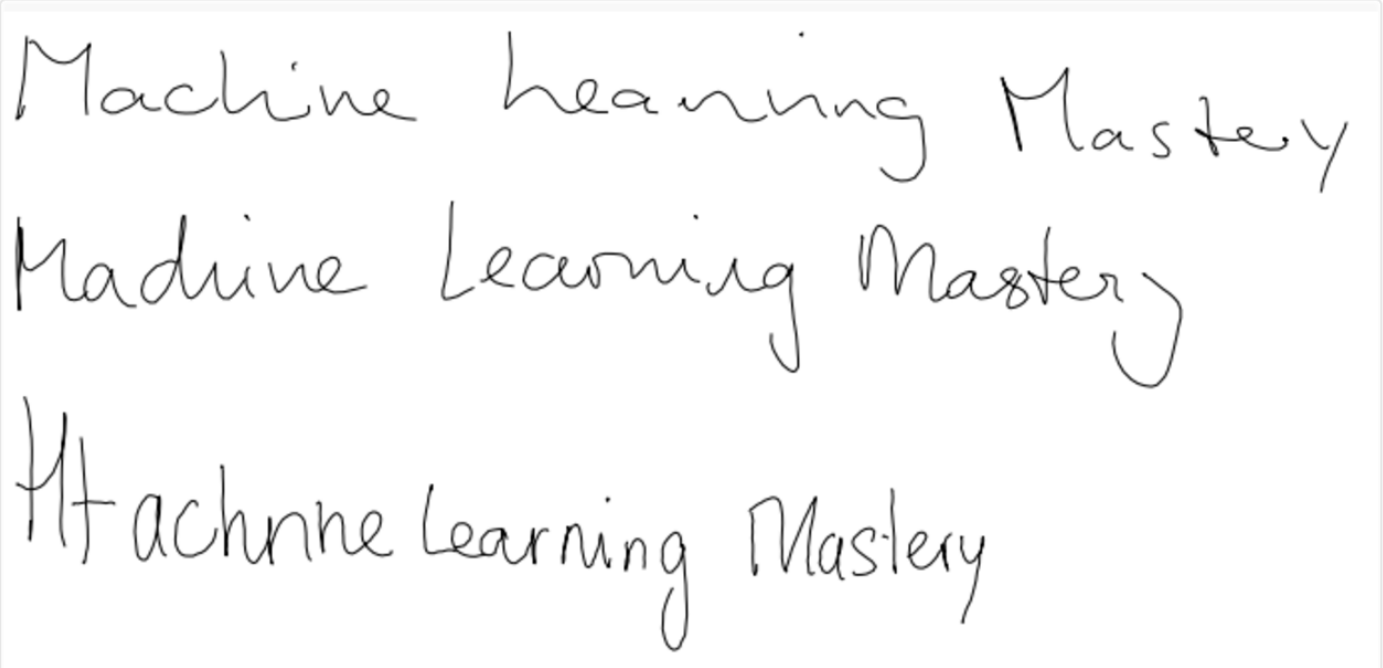
Example of instant visual translation, taken from the Google Blog.

Applications: Object Classification in Photographs

Example of Object Classification

Taken from ImageNet Classification with Deep Convolutional Neural Networks

Applications: Automatic Handwriting Generation



Sample of Automatic Handwriting Generation

Applications: Image Caption Generation

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

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

Automatic Image Caption Generation
Sample taken from Andrej Karpathy, Li Fei-Fei

Applications: Automatic Game Playing

- This is a task where a model learns how to play a computer game based only on the pixels on the screen.

Applications: Automatic Game Playing

- This is a task where a model learns how to play a computer game based only on the pixels on the screen.
- This very difficult task is the domain of deep reinforcement models and is the breakthrough that DeepMind (now part of google) is renown for achieving.

Applications: Automatic Game Playing

- This is a task where a model learns how to play a computer game based only on the pixels on the screen.
- This very difficult task is the domain of deep reinforcement models and is the breakthrough that DeepMind (now part of google) is renown for achieving.
- https://www.youtube.com/watch?v=5O5oUTrFJuQ

Feed Forward Networks

- Many applications of deep learning use feedforward neural network architectures

Feed Forward Networks

- Many applications of deep learning use feedforward neural network architectures
- Which learn to map a fixed-size input (for example, an image) to a fixed-size output

Feed Forward Networks

- Many applications of deep learning use feedforward neural network architectures
- Which learn to map a fixed-size input (for example, an image) to a fixed-size output
- To go from one layer to the next, a set of units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear function.

Feed Forward Networks

- Many applications of deep learning use feedforward neural network architectures
- Which learn to map a fixed-size input (for example, an image) to a fixed-size output
- To go from one layer to the next, a set of units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear function.
- At present, the most popular (faster) non-linear function is the rectified linear unit (ReLU), which is simply the half-wave rectifier $f(z) = max(z, 0)$.

Feed Forward Networks

- Many applications of deep learning use feedforward neural network architectures
- Which learn to map a fixed-size input (for example, an image) to a fixed-size output
- To go from one layer to the next, a set of units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear function.
- At present, the most popular (faster) non-linear function is the rectified linear unit (ReLU), which is simply the half-wave rectifier $f(z) = max(z, 0)$.
- In past decades, neural nets used smoother non-linearities, such as $tanh(z)$ or $1/(1 + exp(-z))$,

Winter AI

- In the late 1990s, neural nets and backpropagation were largely forsaken by the machine-learning community and ignored by the computer-vision and speech-recognition communities.

Winter AI

- In the late 1990s, neural nets and backpropagation were largely forsaken by the machine-learning community and ignored by the computer-vision and speech-recognition communities.
- It was widely thought that learning useful, multistage, feature extractors with little prior knowledge was infeasible.

Winter AI

- In the late 1990s, neural nets and backpropagation were largely forsaken by the machine-learning community and ignored by the computer-vision and speech-recognition communities.
- It was widely thought that learning useful, multistage, feature extractors with little prior knowledge was infeasible.
- In particular, it was commonly thought that simple gradient descent would get trapped in poor local minima

Winter AI

- In the late 1990s, neural nets and backpropagation were largely forsaken by the machine-learning community and ignored by the computer-vision and speech-recognition communities.
- It was widely thought that learning useful, multistage, feature extractors with little prior knowledge was infeasible.
- In particular, it was commonly thought that simple gradient descent would get trapped in poor local minima
- weight configurations for which no small change would reduce the average error.

Winter AI

- In practice, poor local minima are rarely a problem with large networks.

Winter AI

- In practice, poor local minima are rarely a problem with large networks.
- Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality.

Winter AI

- In practice, poor local minima are rarely a problem with large networks.
- Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality.
- Instead, the landscape is packed with a combinatorially large number of saddle points where the gradient is zero, and the surface curves up in most dimensions and curves down in the remainder.

Winter AI

- In practice, poor local minima are rarely a problem with large networks.
- Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality.
- Instead, the landscape is packed with a combinatorially large number of saddle points where the gradient is zero, and the surface curves up in most dimensions and curves down in the remainder.
- Hence, it does not much matter which of these saddle points the algorithm gets stuck at.

Neural Networks Resurgence

- Interest in deep feedforward network was revived around 2006 (CIFAR, Canada)

Neural Networks Resurgence

- Interest in deep feedforward network was revived around 2006 (CIFAR, Canada)
- Introduced unsupervised learning procedures that could create layers of feature detectors without requiring labelled data

Neural Networks Resurgence

- Interest in deep feedforward network was revived around 2006 (CIFAR, Canada)
- Introduced unsupervised learning procedures that could create layers of feature detectors without requiring labelled data
- The objective in learning each layer of feature detectors was to be able to reconstruct or model the activities of feature detectors (or raw inputs) in the layer below

Pre-training

- By ‘pre-training’ several layers of progressively more complex feature detectors, the weights of a deep network could be initialized to sensible values

Pre-training

- By 'pre-training' several layers of progressively more complex feature detectors, the weights of a deep network could be initialized to sensible values
- A final layer of output units could be then added to the top of the network and the whole deep system could be fine-tuned using standard backpropagation

Pre-training

- By 'pre-training' several layers of progressively more complex feature detectors, the weights of a deep network could be initialized to sensible values
- A final layer of output units could be then added to the top of the network and the whole deep system could be fine-tuned using standard backpropagation
- This worked remarkably well for recognizing handwritten digits or for detecting pedestrians

Pre-training

- By 'pre-training' several layers of progressively more complex feature detectors, the weights of a deep network could be initialized to sensible values
- A final layer of output units could be then added to the top of the network and the whole deep system could be fine-tuned using standard backpropagation
- This worked remarkably well for recognizing handwritten digits or for detecting pedestrians
- Especially when the amount of labelled data was very limited.

First Outstanding Results

- 2009, the first major application of this pre-training approach was in speech recognition (GPU)

First Outstanding Results

- 2009, the first major application of this pre-training approach was in speech recognition (GPU)
- It achieved record-breaking results on a standard speech recognition benchmark

First Outstanding Results

- 2009, the first major application of this pre-training approach was in speech recognition (GPU)
- It achieved record-breaking results on a standard speech recognition benchmark
- 2012, versions of the deep net from 2009 were being developed by many of the major speech groups

First Outstanding Results

- 2009, the first major application of this pre-training approach was in speech recognition (GPU)
- It achieved record-breaking results on a standard speech recognition benchmark
- 2012, versions of the deep net from 2009 were being developed by many of the major speech groups
- leading to significantly better generalization when the number of labelled examples is small

Convolutional Neural Networks Resurgence

- A particular type of deep, feedforward network was much easier to train and generalized much better than networks with full connectivity

Convolutional Neural Networks Resurgence

- A particular type of deep, feedforward network was much easier to train and generalized much better than networks with full connectivity
- Convolutional neural networks has recently been widely adopted by the computer vision community

CNN Intuition

- CNNs are designed to process data that come in the form of multiple arrays

CNN Intuition

- CNNs are designed to process data that come in the form of multiple arrays
- A color image composed of three 2D arrays containing pixel intensities in the three colour channels

CNN Intuition

- CNNs are designed to process data that come in the form of multiple arrays
- A color image composed of three 2D arrays containing pixel intensities in the three colour channels
- Many data modalities are in the form of multiple arrays: 1D for signal and sequences, including languages

CNN Intuition

- CNNs are designed to process data that come in the form of multiple arrays
- A color image composed of three 2D arrays containing pixel intensities in the three colour channels
- Many data modalities are in the form of multiple arrays: 1D for signal and sequences, including languages
- 2D for images or audio spectrograms and 3D for video or volumetric images

Four key ideas behind CNNs

- local connections

Four key ideas behind CNNs

- local connections
- shared weights

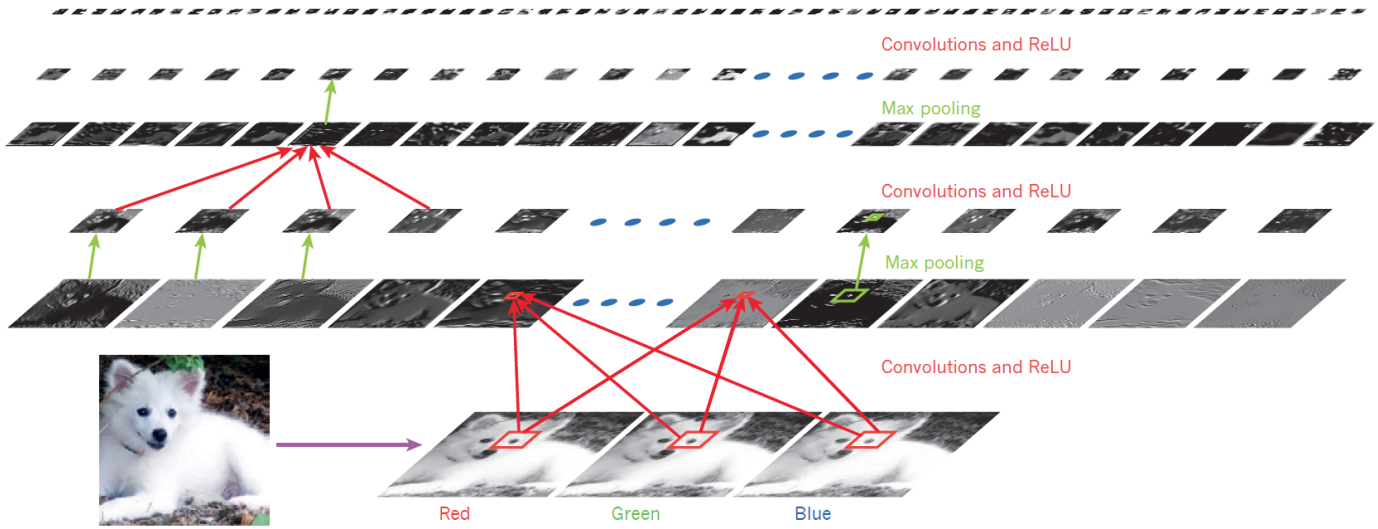
Four key ideas behind CNNs

- local connections
- shared weights
- pooling

Four key ideas behind CNNs

- local connections
- shared weights
- pooling
- use of many layers

Architecture of a typical CNN



ImageNet

- CNNs were largely forsaken buy the mainstream computer vision and ML communities until the ImageNet competition in 2012.

ImageNet

- CNNs were largely forsaken buy the mainstream computer vision and ML communities until the ImageNet competition in 2012.
- DNN were applied to a data set of about a million images from that web that contained 1,000 different clases

ImageNet

- CNNs were largely forsaken buy the mainstream computer vision and ML communities until the ImageNet competition in 2012.
- DNN were applied to a data set of about a million images from that web that contained 1,000 different clases
- Spectacular results were achieved: almost halving the error rates of the best competing approaches

ImageNet

- CNNs were largely forsaken buy the mainstream computer vision and ML communities until the ImageNet competition in 2012.
- DNN were applied to a data set of about a million images from that web that contained 1,000 different clases
- Spectacular results were achieved: almost halving the error rates of the best competing approaches
- This success came from the efficient use of GPUs, ReLUs, dropout (regularization technique) and techniques to generate more training examples

ImageNet

ImageNet 2010

Locality constrained linear coding + SVM	NEC & UIUC
Fisher kernel + SVM	Xerox Research Center Europe
SIFT features + LI2C	Nanyang Technological Institute
SIFT features + k-Nearest Neighbors	Laboratoire d'Informatique de Grenoble
Color features + canonical correlation analysis	National Institute of Informatics, Tokyo

ImageNet 2011

Compressed Fisher kernel + SVM	Xerox Research Center Europe
SIFT bag-of-words + VQ + SVM	University of Amsterdam & University of
SIFT + ?	ISI Lab, Tokyo University

ImageNet 2012

Deep convolutional neural network	University of Toronto
Discriminatively trained DPMs	University of Oxford
Fisher-based SIFT features + SVM	ISI Lab, Tokyo University

Computer Vision Revolution

- Computer vision revolution: CNNs are the dominant approach for almost all recognition and detection tasks and approach human performance on some tasks

Computer Vision Revolution

- Computer vision revolution: CNNs are the dominant approach for almost all recognition and detection tasks and approach human performance on some tasks
- Recent CNNs architectures have 10 to 20 layers of ReLUs, hundred of millions of weights, and billions of connection between units.

Computer Vision Revolution

- Computer vision revolution: CNNs are the dominant approach for almost all recognition and detection tasks and approach human performance on some tasks
- Recent CNNs architectures have 10 to 20 layers of ReLUs, hundred of millions of weights, and billions of connection between units.
- whereas training have taken weeks two years ago, progress in hardware, software and parallelization have reduced training times to a few hours

Performance

- The performance of CNN based vision systems has caused most major technology companies, including Google, Facebook, Microsoft, Yahoo, Twitter and Adobe, as well as a quickly growing number of startups to initiate research and development projects and to deploy CNN based image understanding products and services

Performance

- The performance of CNN based vision systems has caused most major technology companies, including Google, Facebook, Microsoft, Yahoo, Twitter and Adobe, as well as a quickly growing number of startups to initiate research and development projects and to deploy CNN based image understanding products and services
- CNNs are easily amenable to efficient hardware implementations in chips of field programmable gate arrays.

Performance

- The performance of CNN based vision systems has caused most major technology companies, including Google, Facebook, Microsoft, Yahoo, Twitter and Adobe, as well as a quickly growing number of startups to initiate research and development projects and to deploy CNN based image understanding products and services
- CNNs are easily amenable to efficient hardware implementations in chips of field programmable gate arrays.
- NVIDIA, Intel, Qualcomm and Samsung are developing CNN chips to enable real time vision applications in smartphones, cameras, robots and self-driving cars

CNNs

- CNNs are very similar to ordinary Neural Networks

CNNs

- CNNs are very similar to ordinary Neural Networks
- They are made up of neurons that have learnable weights and biases.

CNNs

- CNNs are very similar to ordinary Neural Networks
- They are made up of neurons that have learnable weights and biases.
- Each neuron receieves som inputs, performs a dot product and optionally follows it with a non-linearity.

CNNs

- CNNs are very similar to ordinary Neural Networks
- They are made up of neurons that have learnable weights and biases.
- Each neuron receieves som inputs, performs a dot product and optionally follows it with a non-linearity.
- The whole network still expresses a single differentiable score function

CNNs

- CNNs are very similar to ordinary Neural Networks
- They are made up of neurons that have learnable weights and biases.
- Each neuron receieves som inputs, performs a dot product and optionally follows it with a non-linearity.
- The whole network still expresses a single differentiable score function
- They still have a loss function o the last fully-connected layer

What does change?

- ConvNet architectures make the explicit assumption that the inputs are images

What does change?

- ConvNet architectures make the explicit assumption that the inputs are images
- Which allow us to encode certain properties into the architecture

What does change?

- ConvNet architectures make the explicit assumption that the inputs are images
- Which allow us to encode certain properties into the architecture
- These the make the forward function more efficient to implement and

What does change?

- ConvNet architectures make the explicit assumption that the inputs are images
- Which allow us to encode certain properties into the architecture
- These the make the forward function more efficient to implement and
- vastly reduce the amount of parameters in the network

Regular Neural Nets

- NN receive an input (a single vector) and transform it through a series of hidden layers

Regular Neural Nets

- NN receive an input (a single vector) and transform it through a series of hidden layers
- Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer

Regular Neural Nets

- NN receive an input (a single vector) and transform it through a series of hidden layers
- Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer
- Neuron in a single layer function completely independently do not share any connections

Regular Neural Nets

- NN receive an input (a single vector) and transform it through a series of hidden layers
- Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer
- Neuron in a single layer function completely independently do not share any connections
- The last fully-connected layer, in classification settings, it represents the class scores

Regular Neural Nets

- NN receive an input (a single vector) and transform it through a series of hidden layers
- Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer
- Neuron in a single layer function completely independently do not share any connections
- The last fully-connected layer, in classification settings, it represents the class scores
- Regular NN don't scalle well to full images

The Problem with NN

- In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32, high, 3 color channels)

The Problem with NN

- In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32, high, 3 color channels)
- A single fully-connected neuron in a first hidden layer would have $32 * 32 * 3 = 3072$ weights

The Problem with NN

- In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32, high, 3 color channels)
- A single fully-connected neuron in a first hidden layer would have $32 * 32 * 3 = 3072$ weights
- This fully-connectd structure does not scale to larger images

The Problem with NN

- In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32, high, 3 color channels)
- A single fully-connected neuron in a first hidden layer would have $32 * 32 * 3 = 3072$ weights
- This fully-connectd structure does not scale to larger images
- An image of $200 \times 200 \times 3$, would lead to neurons that have $200 * 200 * 3 = 120000$ wights

The Problem with NN

- In CIFAR-10, images are only of size $32 \times 32 \times 3$ (32 wide, 32, high, 3 color channels)
- A single fully-connected neuron in a first hidden layer would have $32 * 32 * 3 = 3072$ weights
- This fully-connectd structure does not scale to larger images
- An image of $200 \times 200 \times 3$, would lead to neurons that have $200 * 200 * 3 = 120000$ wights
- This full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting

3D Volumes of Neurons

- CNNs take advantange of the fact that the input consists of images and they constrain the architecture in a more sensible way

3D Volumes of Neurons

- CNNs take advantange of the fact that the input consists of images and they constrain the architecture in a more sensible way
- Layers of ConvNets have neurons arranged in 3 dimensions: width, height, depth

3D Volumes of Neurons

- CNNs take advantange of the fact that the input consists of images and they constrain the architecture in a more sensible way
- Layers of ConvNets have neurons arranged in 3 dimensions: width, height, depth
- The input images in CIFAR-10 are an input volume of activations and the volumen has dimensions $(32 \times 32 \times 3)$

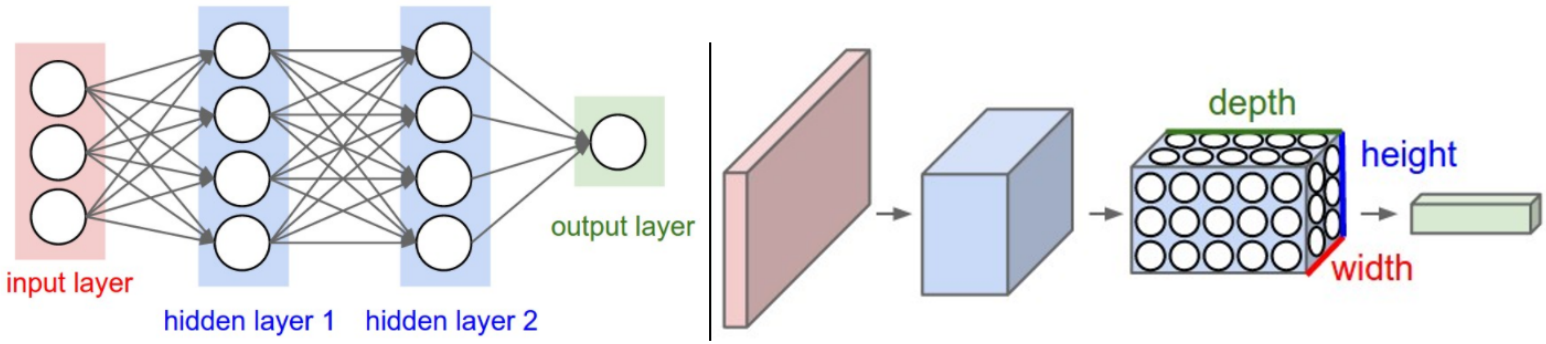
3D Volumes of Neurons

- CNNs take advantange of the fact that the input consists of images and they constrain the architecture in a more sensible way
- Layers of ConvNets have neurons arranged in 3 dimensions: width, height, depth
- The input images in CIFAR-10 are an input volume of activations and the volumen has dimensions $(32 \times 32 \times 3)$
- The neurons in a layer will only be connected to a small region of the layer before it

3D Volumes of Neurons

- CNNs take advantange of the fact that the input consists of images and they constrain the architecture in a more sensible way
- Layers of ConvNets have neurons arranged in 3 dimensions: width, height, depth
- The input images in CIFAR-10 are an input volume of activations and the volumen has dimensions $(32 \times 32 \times 3)$
- The neurons in a layer will only be connected to a small region of the layer before it
- The final output layer would for CIFAR-10 have dimensions $1 \times 1 \times 10$, because we will reduce the full image into a single vector of class scores

3D Volumes of Neurons



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Layers Used to Build ConvNets

a ConvNet transforms one volume of activations to another through a differentiable function.
We use three main types of layers:

- Convolutional Layer

Layers Used to Build ConvNets

a ConvNet transforms one volume of activations to another through a differentiable function.
We use three main types of layers:

- Convolutional Layer
- Pooling Layer

Layers Used to Build ConvNets

a ConvNet transforms one volume of activations to another through a differentiable function.
We use three main types of layers:

- Convolutional Layer
- Pooling Layer
- Fully-Connected Layer

Example Architecture

For CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]:

- INPUT $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case an image of width 32, height 32 and with three color channels R,G,B

Example Architecture

For CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]:

- INPUT $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case an image of width 32, height 32 and with three color channels R,G,B
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume

Example Architecture

For CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]:

- INPUT $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case an image of width 32, height 32 and with three color channels R,G,B
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume
- This may results in volume such as $[32 \times 32 \times 12]$ if we decided to use 12 filters

Example Architecture

- RELU layer will apply an elementwise activation funcion, such as the $max(0, x)$. This leaves the size of the volume unchanged

Example Architecture

- RELU layer will apply an elementwise activation funcion, such as the $max(0, x)$. This leaves the size of the volume unchanged
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as $[16 \times 16 \times 12]$

Example Architecture

- RELU layer will apply an elementwise activation funcion, such as the $max(0, x)$. This leaves the size of the volume unchanged
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as $[16 \times 16 \times 12]$
- FC (fully-connected) layer will compute the class scores, resulting in volume of size $[1 \times 1 \times 10]$, where each of the 10 numbers correspond to a class score. Each neuron in this layer will be connected to all the numbers in the previous volume

Example Architecture

- ConvNets transform the original image layer by layer from the original pixel values to the final class scores.

Example Architecture

- ConvNets transform the original image layer by layer from the original pixel values to the final class scores.
- some layers contain parameters and other don't.

Example Architecture

- ConvNets transform the original image layer by layer from the original pixel values to the final class scores.
- some layers contain parameters and other don't.
- In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons).

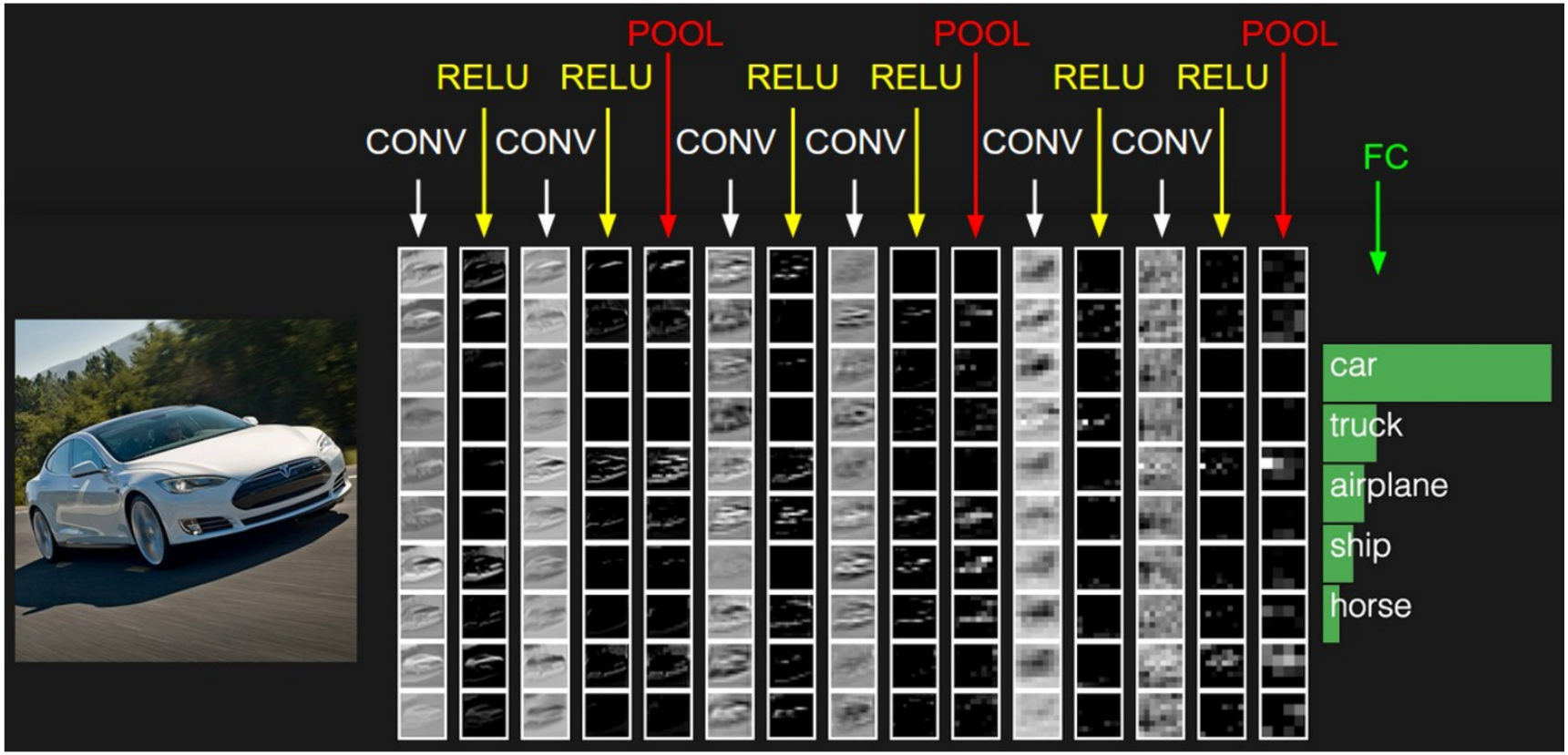
Example Architecture

- ConvNets transform the original image layer by layer from the original pixel values to the final class scores.
- some layers contain parameters and other don't.
- In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons).
- On the other hand, the RELU/POOL layers will implement a fixed function.

Example Architecture

- ConvNets transform the original image layer by layer from the original pixel values to the final class scores.
- some layers contain parameters and other don't.
- In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons).
- On the other hand, the RELU/POOL layers will implement a fixed function.
- The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

Architecture



Architecture Summary

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)

Architecture Summary

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)

Architecture Summary

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function

Architecture Summary

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don?t)

Architecture Summary

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don?t)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn?t)

Convolutional Layer

- The CONV layer's parameters consist of a set of learnable filters.

Convolutional Layer

- The CONV layer's parameters consist of a set of learnable filters.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume

Convolutional Layer

- The CONV layer's parameters consist of a set of learnable filters.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume
- Example a typical filter on a first layer might have size $5 \times 5 \times 3$

Convolutional Layer

- The CONV layer's parameters consist of a set of learnable filters.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume
- Example a typical filter on a first layer might have size $5 \times 5 \times 3$
- During the forward pass, we slide (convolve) each filter across the width and height of the input volume and

Convolutional Layer

- The CONV layer's parameters consist of a set of learnable filters.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume
- Example a typical filter on a first layer might have size $5 \times 5 \times 3$
- During the forward pass, we slide (convolve) each filter across the width and height of the input volume and
- compute dot products between the entries of the filter and the input at any position

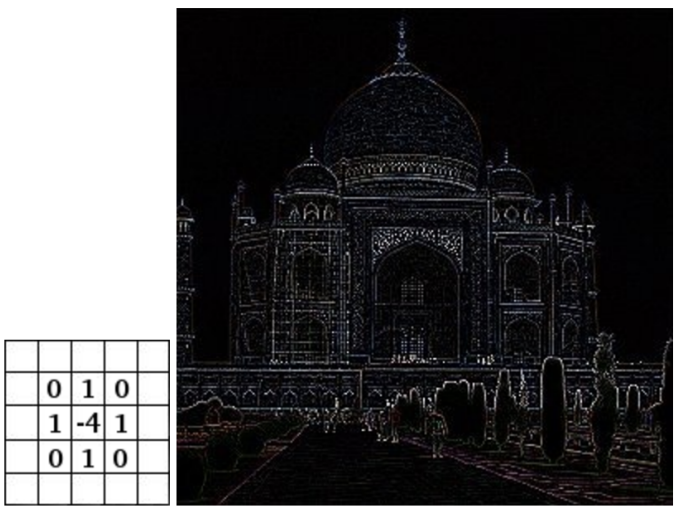
Intuitions About Convolutions

Averaging Each Pixel with its neighboring values blurs an image

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

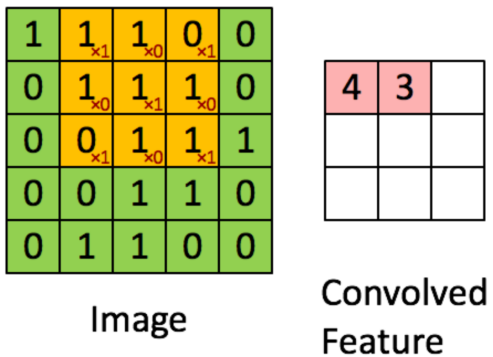
Intuitions About Convolutions

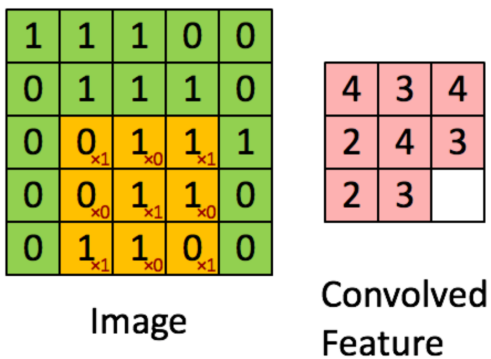
Taking the difference between a pixel and its neighbors detect edges



What is a Convolution?

the easiest way to understand is by thinking of it as a sliding window function applied to a matrix





Convolutional Layer

- As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position

Convolutional Layer

- As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position
- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer

Convolutional Layer

- As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position
- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer
- We will have an entire set of filters in each CONV layer, and each of them will produce a separate 2-dimensional activation map

Convolutional Layer

- As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position
- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer
- We will have an entire set of filters in each CONV layer, and each of them will produce a separate 2-dimensional activation map
- We will stack these activation maps along the depth dimension and produce the output volume

Local Connectivity

- When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume

Local Connectivity

- When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume
- Instead, we will connect each neuron to only a local region of the input volume

Local Connectivity

- When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume
- Instead, we will connect each neuron to only a local region of the input volume
- The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (the filter size)

Local Connectivity

- When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume
- Instead, we will connect each neuron to only a local region of the input volume
- The spatial extent of this connectivity is a hyperparameter called the receptive field of the neuron (the filter size)
- The extent of the connectivity along the depth axis is always equal to the depth of the input volume.

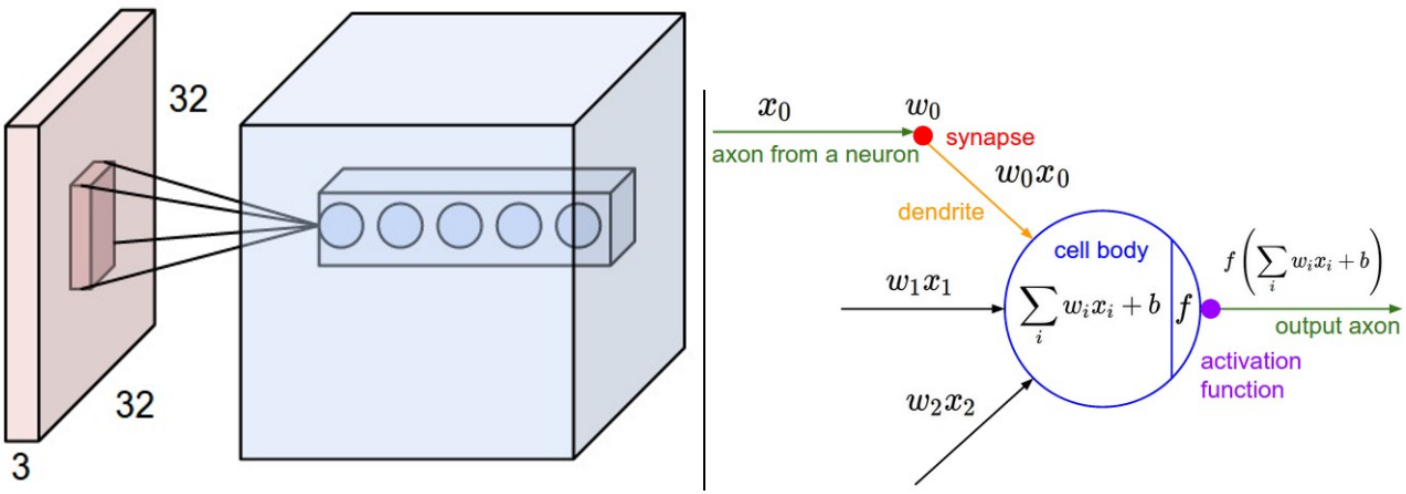
Example

Suppose that the input volume has size $[32 \times 32 \times 3]$
If the receptive filed (filter size) is 5×5, then each neuron in the Conv Layer will have weights to a $[5 \times 5 \times 3]$ region in the input volume, for a total of $5 * 5 * 3 = 75$ weights (and $+1$ bias parameters)
Notice that the extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume

Example II

Suppose an input volume had size $[16 \times 16 \times 20]$.
Then using an example receptive field size of 3×3, every neuron in the Conv Layer would now have a total $3 * 3 * 20 = 180$ connections to the input volume.

Local


32
32
3
x_0
w_0
synapse
axon from a neuron
w_0x_0
dendrite
cell body
w_1x_1
$\sum_i w_ix_i + b$
f
$f\left(\sum_i w_ix_i + b\right)$
output axon
activation function
w_2x_2

Spatial Arrangement

Three hyperparameters control the size of the output volume:

- Depth

Spatial Arrangement

Three hyperparameters control the size of the output volume:

- Depth
- Stride

Spatial Arrangement

Three hyperparameters control the size of the output volume:

- Depth
- Stride
- Zero-padding

Depth

- It corresponds to the number of filters we would like to use, each learning to look for something different in the input

Depth

- It corresponds to the number of filters we would like to use, each learning to look for something different in the input
- If the first Conv Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color

Depth

- It corresponds to the number of filters we would like to use, each learning to look for something different in the input
- If the first Conv Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color
- We will refer to a set of neurons that are all looking at the same region of the input as a depth column

Stride

- We must specify the stride with which we slide the filter

Stride

- We must specify the stride with which we slide the filter
- When the stride is 1 then we move the filters one pixel at a time

Stride

- We must specify the stride with which we slide the filter
- When the stride is 1 then we move the filters one pixel at a time
- When the stride is 2 then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes

Zero-padding

- Sometimes it will be convenient to pad the input volume with zeros around the border

Zero-padding

- Sometimes it will be convenient to pad the input volume with zeros around the border
- The size of this zero-padding is a hyperparameter

Zero-padding

- Sometimes it will be convenient to pad the input volume with zeros around the border
- The size of this zero-padding is a hyperparameter
- It will allow us to control the spatial size of the output volumes (e.g. to preserve the spatial size)

Formula

We can compute the spatial size of the output volume as a function of

- The input volume size (W)

Formula

We can compute the spatial size of the output volume as a function of

- The input volume size (W)
- The receptive field size of the Conv Layer neurons (F)

Formula

We can compute the spatial size of the output volume as a function of

- The input volume size (W)
- The receptive field size of the Conv Layer neurons (F)
- The stride with which they are applied (S)

Formula

We can compute the spatial size of the output volume as a function of

- The input volume size (W)
- The receptive field size of the Conv Layer neurons (F)
- The stride with which they are applied (S)
- The amount of zero padding used (P) on the border

Formula

We can compute the spatial size of the output volume as a function of

- The input volume size (W)
- The receptive field size of the Conv Layer neurons (F)
- The stride with which they are applied (S)
- The amount of zero padding used (P) on the border
- How many neurons fit is given by: $(W - F + 2P)/S + 1$

Examples

$(W - F + 2P)/S + 1$

- For example for a 7×7 input and a 3×3 filter with stride 1 and pad 0 we would get a 5×5 output

Examples

$(W - F + 2P)/S + 1$

- For example for a 7×7 input and a 3×3 filter with stride 1 and pad 0 we would get a 5×5 output
- With stride 2

Examples

$(W - F + 2P)/S + 1$

- For example for a 7×7 input and a 3×3 filter with stride 1 and pad 0 we would get a 5×5 output
- With stride 2
- We would get a 3×3 output

Examples

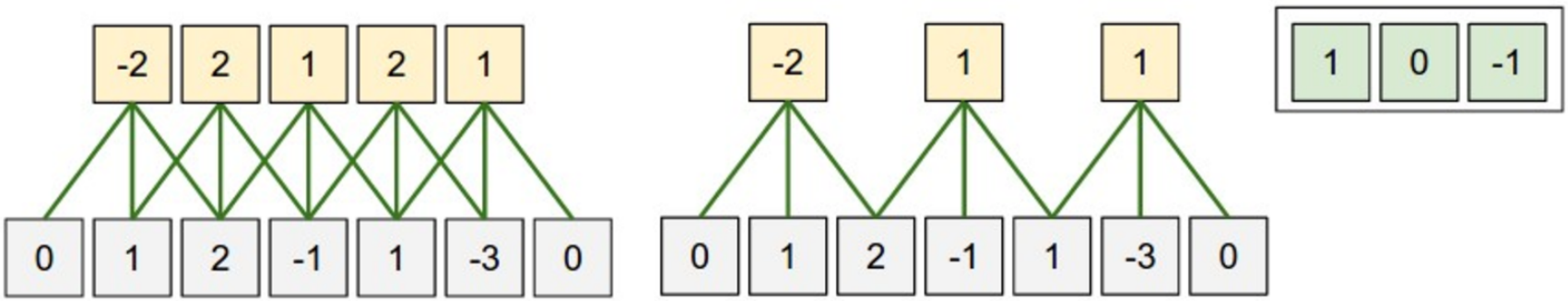


Illustration of spatial arrangement. In this example there is only one spatial dimension (x-axis), one neuron with a receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1. **Left:** The neuron strided across the input in stride of S = 1, giving output of size (5 - 3 + 2)/1+1 = 5. **Right:** The neuron uses stride of S = 2, giving output of size (5 - 3 + 2)/2+1 = 3. Notice that stride S = 3 could not be used since it wouldn't fit neatly across the volume. In terms of the equation, this can be determined since (5 - 3 + 2) = 4 is not divisible by 3.

The neuron weights are in this example [1,0,-1] (shown on very right), and its bias is zero. These weights are shared across all yellow neurons (see parameter sharing below).

Use of Zero-padding

- In the figure on left, note that the input dimension was 5 and the output dimension was equal: also 5.

Use of Zero-padding

- In the figure on left, note that the input dimension was 5 and the output dimension was equal: also 5.
- This worked out so because our receptive fields were 3 and we used zero padding of 1

Use of Zero-padding

- In the figure on left, note that the input dimension was 5 and the output dimension was equal: also 5.
- This worked out so because our receptive fields were 3 and we used zero padding of 1
- If there was no zero-padding used, then the output volume would have had spatial dimension of 3

Use of Zero-padding

- In the figure on left, note that the input dimension was 5 and the output dimension was equal: also 5.
- This worked out so because our receptive fields were 3 and we used zero padding of 1
- If there was no zero-padding used, then the output volume would have had spatial dimension of 3
- In general, setting zero padding to be $P = (F - 1)/2$ when the stride is $S = 1$ ensures that the input volume and output volume will have the same size spatially.

Constraints on Strides

- Note that the spatial arrangement hyperparameters have mutual constraints.

Constraints on Strides

- Note that the spatial arrangement hyperparameters have mutual constraints.
- For example, when the input has size $W = 10$, no zero-padding is used $P = 0$, and the filter size is $F = 3$, then

Constraints on Strides

- Note that the spatial arrangement hyperparameters have mutual constraints.
- For example, when the input has size $W = 10$, no zero-padding is used $P = 0$, and the filter size is $F = 3$, then
- It would be impossible to use stride $S = 2$, since $(W - F + 2P)/S + 1 = (10 - 3 + 0)/2 + 1 = 4.5$,

Constraints on Strides

- Note that the spatial arrangement hyperparameters have mutual constraints.
- For example, when the input has size $W = 10$, no zero-padding is used $P = 0$, and the filter size is $F = 3$, then
- It would be impossible to use stride $S = 2$, since $(W - F + 2P)/S + 1 = (10 - 3 + 0)/2 + 1 = 4.5$,
- Not an integer, indicating that the neurons don?t fit neatly and symmetrically across the input.

Real World Example

- The Krizhevsky et al architecture that won the ImageNet challenge in 2012 accepted images of size $[227 \times 227 \times 3]$

Real World Example

- The Krizhevsky et al architecture that won the ImageNet challenge in 2012 accepted images of size $[227 \times 227 \times 3]$
- The first Conv Layer, it used neurons with receptive field size $F = 11$, stride $S = 4$ and no zero-padding $P = 0$.

Real World Example

- The Krizhevsky et al architecture that won the ImageNet challenge in 2012 accepted images of size $[227 \times 227 \times 3]$
- The first Conv Layer, it used neurons with receptive field size $F = 11$, stride $S = 4$ and no zero-padding $P = 0$.
- Since $(227 - 11)/4 + 1 = 55$, and since the Conv Layer had a depth of $K = 96$

Real World Example

- The Krizhevsky et al architecture that won the ImageNet challenge in 2012 accepted images of size $[227 \times 227 \times 3]$
- The first Conv Layer, it used neurons with receptive field size $F = 11$, stride $S = 4$ and no zero-padding $P = 0$.
- Since $(227 - 11)/4 + 1 = 55$, and since the Conv Layer had a depth of $K = 96$
- The Conv layer output volume had size $[55 \times 55 \times 96]$

Real World Example

- The Krizhevsky et al architecture that won the ImageNet challenge in 2012 accepted images of size $[227 \times 227 \times 3]$
- The first Conv Layer, it used neurons with receptive field size $F = 11$, stride $S = 4$ and no zero-padding $P = 0$.
- Since $(227 - 11)/4 + 1 = 55$, and since the Conv Layer had a depth of $K = 96$
- The Conv layer output volume had size $[55 \times 55 \times 96]$
- Each of the $55 * 55 * 96$ neurons in this volume was connected to a region of size $[11 \times 11 \times 3]$ in the input volume

Parameter Sharing

- It is used to control the number of parameters

Parameter Sharing

- It is used to control the number of parameters
- In the previous example, there are $55 * 55 * 96 = 290400$ neurons in the first Conv Layer, and each has $11 * 11 * 3 = 363$ weights and 1 bias

Parameter Sharing

- It is used to control the number of parameters
- In the previous example, there are $55 * 55 * 96 = 290400$ neurons in the first Conv Layer, and each has $11 * 11 * 3 = 363$ weights and 1 bias
- Together, this adds up to $290400 * 364 = 105,705600$ parameters on the first layer alone

Parameter Sharing

- It is used to control the number of parameters
- In the previous example, there are $55 * 55 * 96 = 290400$ neurons in the first Conv Layer, and each has $11 * 11 * 3 = 363$ weights and 1 bias
- Together, this adds up to $290400 * 364 = 105,705600$ parameters on the first layer alone
- Clearly, this number is very high

Parameter Reduction

- We can dramatically reduce the number of parameters by making one reasonable assumption

Parameter Reduction

- We can dramatically reduce the number of parameters by making one reasonable assumption
- That if one feature is useful to compute at some spatial position (x, y), the it should also be useful to compute at a different position $(x2, y2)$

Parameter Reduction

- We can dramatically reduce the number of parameters by making one reasonable assumption
- That if one feature is useful to compute at some spatial position (x, y), the it should also be useful to compute at a different position $(x2, y2)$
- In other words, denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size $[55 \times 55 \times 96]$ has 96 depth slices, each of size $[55 \times 55]$)

Parameter Reduction

- We can dramatically reduce the number of parameters by making one reasonable assumption
- That if one feature is useful to compute at some spatial position (x, y), the it should also be useful to compute at a different position $(x2, y2)$
- In other words, denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size $[55 \times 55 \times 96]$ has 96 depth slices, each of size $[55 \times 55]$)
- We are going to constrain the neurons in each depth slice to use the same weights and bias.

Parameter Reduction

- We can dramatically reduce the number of parameters by making one reasonable assumption
- That if one feature is useful to compute at some spatial position (x, y), the it should also be useful to compute at a different position $(x2, y2)$
- In other words, denoting a single 2-dimensional slice of depth as a depth slice (e.g. a volume of size $[55 \times 55 \times 96]$ has 96 depth slices, each of size $[55 \times 55]$)
- We are going to constrain the neurons in each depth slice to use the same weights and bias.
- With this parameter sharing scheme, the first Conv Layer would now have only 96 unique set of weights (one for each depth slice), for a total of 96*11*11*3 = 34,848 unique weights, or 34,944 parameters (+96 biases).

Parameter Reduction

- Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters.

Parameter Reduction

- Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters.
- In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

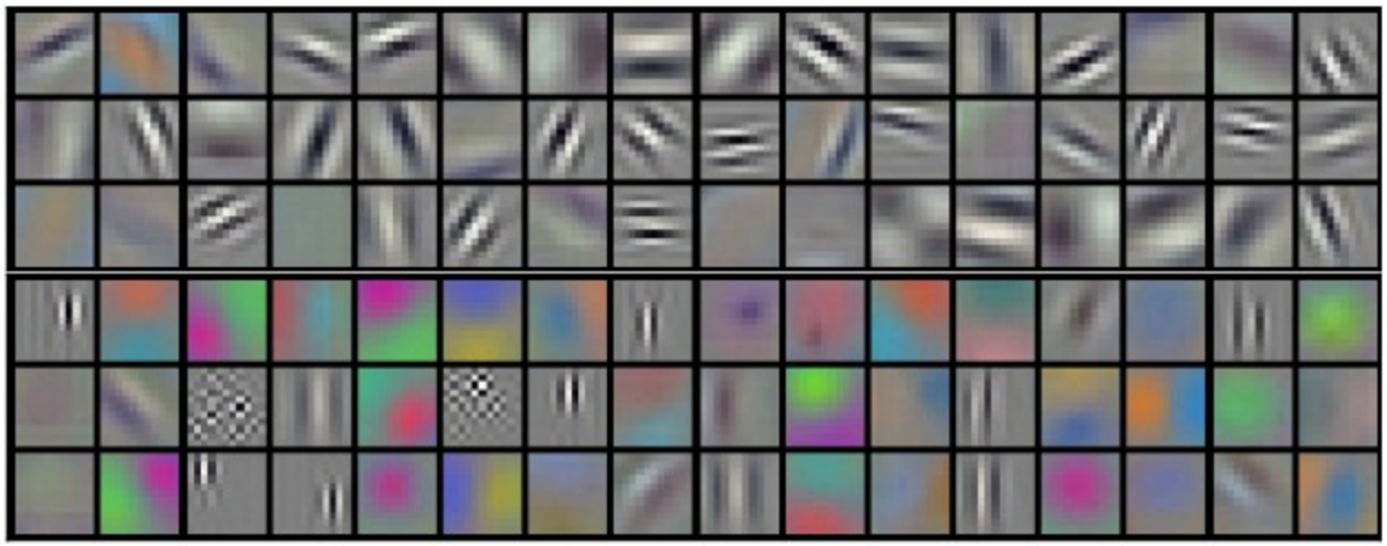
Parameter Reduction

- Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters.
- In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.
- Notice that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a convolution of the neuron?s weights with the input volume

Parameter Reduction

- Alternatively, all 55*55 neurons in each depth slice will now be using the same parameters.
- In practice during backpropagation, every neuron in the volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.
- Notice that if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a convolution of the neuron?s weights with the input volume
- This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input

Parameter Reduction



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

Numpy Examples

- Suppose that the input volume is a numpy array **X**

Numpy Examples

- Suppose that the input volume is a numpy array **X**
- A depth column at position (x, y) would be the activations X[x,y:]

Numpy Examples

- Suppose that the input volume is a numpy array **X**
- A depth column at position (x, y) would be the activations X[x,y:]
- A depth slice, or equivalently an activation map at depth d would be the activations X[:,:,d].

Conv Layer Example

- Suppose that the input volume X has shape X.shape: (11,11,4).

Conv Layer Example

- Suppose that the input volume X has shape X.shape: (11,11,4).
- Suppose further that we use no zero padding ($P = 0$), that the filter size is $F = 5$, and that the stride is $S = 2$.

Conv Layer Example

- Suppose that the input volume X has shape X.shape: (11,11,4).
- Suppose further that we use no zero padding ($P = 0$), that the filter size is $F = 5$, and that the stride is $S = 2$.
- The output volume would therefore have spatial size $(11 - 5)/2 + 1 = 4$, giving a volume with width and height of 4

Conv Layer Example

The activation map in the output volume (call it V), would then look as follows

V[0,0,0] = np.sum(X[:5,:5,:] * W0) + b0

V[1,0,0] = np.sum(X[2:7,:5,:] * W0) + b0

V[2,0,0] = np.sum(X[4:9,:5,:] * W0) + b0

V[3,0,0] = np.sum(X[6:11,:5,:] * W0) + b0

Conv Layer Example

- In numpy, the operation * above denotes elementwise multiplication between the arrays.

Conv Layer Example

- In numpy, the operation * above denotes elementwise multiplication between the arrays.
- The weight vector W0 is the weight vector of that neuron and b0 is the bias.

Conv Layer Example

- In numpy, the operation * above denotes elementwise multiplication between the arrays.
- The weight vector W0 is the weight vector of that neuron and b0 is the bias.
- W0 is assumed to be of shape W0.shape: (5,5,4), since the filter size is 5 and the depth of the input volume is 4.

Conv Layer Example

- In numpy, the operation * above denotes elementwise multiplication between the arrays.
- The weight vector W0 is the weight vector of that neuron and b0 is the bias.
- W0 is assumed to be of shape W0.shape: (5,5,4), since the filter size is 5 and the depth of the input volume is 4.
- At each point, we are computing the dot product as in ordinary neural networks.

Conv Layer Example

- In numpy, the operation * above denotes elementwise multiplication between the arrays.
- The weight vector W0 is the weight vector of that neuron and b0 is the bias.
- W0 is assumed to be of shape W0.shape: (5,5,4), since the filter size is 5 and the depth of the input volume is 4.
- At each point, we are computing the dot product as in ordinary neural networks.
- we are using the same weight and bias (due to parameter sharing), and where the dimensions along the width are increasing in steps of 2 (i.e. the stride).

Conv Layer Example

To construct a second activation map in the output volume, we would have:
V[0,0,1] = np.sum(X[:5,:5,:] * W1) + b1
V[1,0,1] = np.sum(X[2:7,:5,:] * W1) + b1
V[2,0,1] = np.sum(X[4:9,:5,:] * W1) + b1
V[3,0,1] = np.sum(X[6:11,:5,:] * W1) + b1
V[0,1,1] = np.sum(X[:5,2:7,:] * W1) + b1 (example of going along y)

Conv Layer Example

- where we see that we are indexing into the second depth dimension in V (at index 1) because we are computing the second activation map, and that a different set of parameters (W1) is now used.

Conv Layer Example

- where we see that we are indexing into the second depth dimension in V (at index 1) because we are computing the second activation map, and that a different set of parameters (W1) is now used.
- Additionally, recall that these activation maps are often followed elementwise through an activation function such as ReLU.

Summary Conv Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$

Summary Conv Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Summary Conv Layer

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$

Summary Conv Layer

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
  - With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Summary Conv Layer

- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
  - With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
  - In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolution Demo

Conv Demo
http://cs231n.github.io/convolutional-networks/

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.
- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation.
- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.
- Every MAX operation would in this case be taking a max over 4 numbers . The depth dimension remains unchanged.

Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$

Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  their spatial extent F, the stride S,

Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  their spatial extent F, the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:

$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

$$D_2 = D_1$$

Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  their spatial extent F, the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:

$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

$$D_2 = D_1$$

- Introduces zero parameters since it computes a fixed function of the input

Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  their spatial extent F, the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:

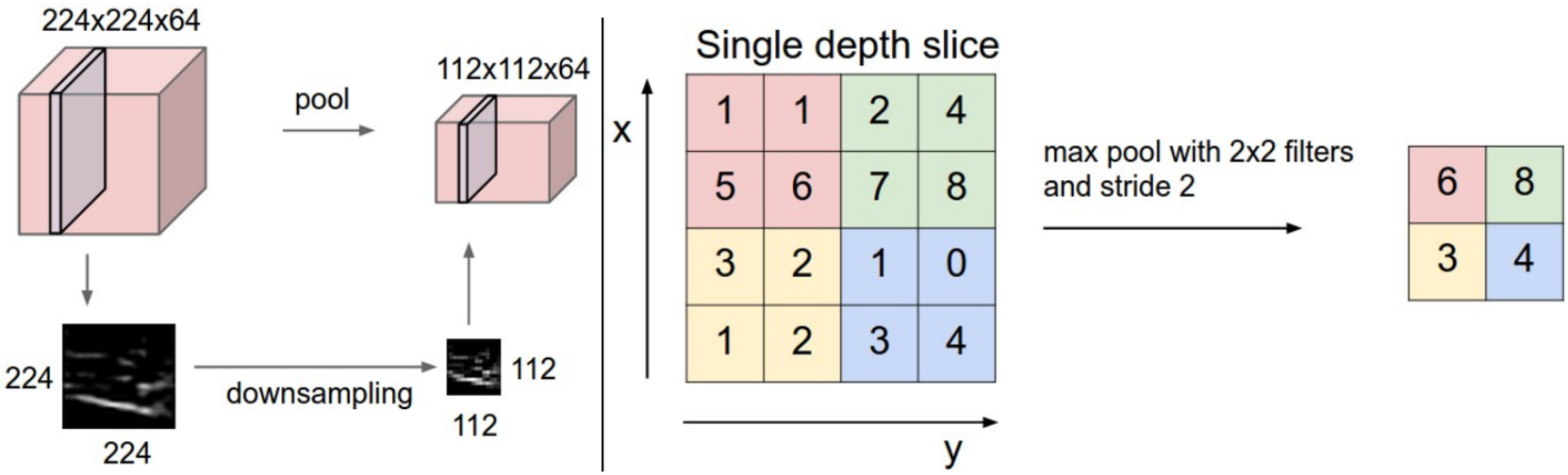
$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

$$D_2 = D_1$$

- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Pooling



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Getting rid of pooling

- Many people dislike the pooling operation and think that we can get away without it

Getting rid of pooling

- Many people dislike the pooling operation and think that we can get away without it
- discard the pooling layer in favor of architecture that only consists of repeated CONV layers

Getting rid of pooling

- Many people dislike the pooling operation and think that we can get away without it
- discard the pooling layer in favor of architecture that only consists of repeated CONV layers
- To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.

Getting rid of pooling

- Many people dislike the pooling operation and think that we can get away without it
- discard the pooling layer in favor of architecture that only consists of repeated CONV layers
- To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.
- Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).

Getting rid of pooling

- Many people dislike the pooling operation and think that we can get away without it
- discard the pooling layer in favor of architecture that only consists of repeated CONV layers
- To reduce the size of the representation they suggest using larger stride in CONV layer once in a while.
- Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).
- It seems likely that future architectures will feature very few to no pooling layers.

Fully-connected layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.

Fully-connected layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks.
- Their activations can hence be computed with a matrix multiplication followed by a bias offset.

ConvNet Architectures

- We have seen that Convolutional Networks are commonly made up of only three layer types: CONV, POOL and FC(short for fully-connected).

ConvNet Architectures

- We have seen that Convolutional Networks are commonly made up of only three layer types: CONV, POOL and FC(short for fully-connected).

ConvNet Architectures

- We have seen that Convolutional Networks are commonly made up of only three layer types: CONV, POOL and FC(short for fully-connected).
- We will also explicitly write the RELU activation function as a layer, which applies elementwise non-linearity.

ConvNet Architectures

- We have seen that Convolutional Networks are commonly made up of only three layer types: CONV, POOL and FC(short for fully-connected).
- We will also explicitly write the RELU activation function as a layer, which applies elementwise non-linearity.
- How these are commonly stacked together to form entire ConvNets?

Layer Patterns

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.

Layer Patterns

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers.

Layer Patterns

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers.
- The last fully-connected layer holds the output, such as the class scores.

Layer Patterns

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers.
- The last fully-connected layer holds the output, such as the class scores.
- In other words, the most common ConvNet architecture follows the pattern:
  INPUT $->$ [[CONV $->$ RELU]*N $->$ POOL?]*M $->$
  [FC $->$ RELU]*K $->$ FC

Layer Patterns

- The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size.
- At some point, it is common to transition to fully-connected layers.
- The last fully-connected layer holds the output, such as the class scores.
- In other words, the most common ConvNet architecture follows the pattern:
  INPUT $->$ [[CONV $->$ RELU]*N $->$ POOL?]*M $->$ [FC $->$ RELU]*K $->$ FC
- Where the * indicates repetition, and the POOL? indicates an optional pooling layer. Moreover, $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$).

Layer Patterns II

- In other words, the most common ConvNet architecture follows the pattern:
  INPUT $->$ [[CONV $->$ RELU]*N $->$ POOL?]*M $->$
  [FC $->$ RELU]*K $->$ FC

Layer Patterns II

- In other words, the most common ConvNet architecture follows the pattern:
  INPUT $->$ [[CONV $->$ RELU]*N $->$ POOL?]*M $->$ [FC $->$ RELU]*K $->$ FC
- Where the * indicates repetition, and the POOL? indicates an optional pooling layer. Moreover, $N \geq 0$ (and usually $N \leq 3$), $M \geq 0$, $K \geq 0$ (and usually $K < 3$).

ConvNet Architectures

- INPUT $->$ FC, implements a linear classifier. Here $N = M = K = 0$.

ConvNet Architectures

- INPUT $->$ FC, implements a linear classifier. Here $N = M = K = 0$.
- INPUT $->$ CONV $->$ RELU $->$ FC

ConvNet Architectures

- INPUT $->$ FC, implements a linear classifier. Here $N = M = K = 0$.
- INPUT $->$ CONV $->$ RELU $->$ FC
- INPUT $->$ [CONV $->$ RELU $->$ POOL]*2 $->$ FC $->$ RELU $->$ FC. Here we see that there is a single CONV layer between every POOL layer.

ConvNet Architectures

- INPUT $->$ FC, implements a linear classifier. Here $N = M = K = 0$.
- INPUT $->$ CONV $->$ RELU $->$ FC
- INPUT $->$ [CONV $->$ RELU $->$ POOL]*2 $->$ FC $->$ RELU $->$ FC. Here we see that there is a single CONV layer between every POOL layer.
- INPUT $->$ [CONV $->$ RELU $->$ CONV $->$ RELU $->$ POOL]*3 $->$ [FC $->$ RELU]*2 $->$ FC Here we see two CONV layers stacked before every POOL layer.

ConvNet Architectures

- INPUT $->$ FC, implements a linear classifier. Here $N = M = K = 0$.
- INPUT $->$ CONV $->$ RELU $->$ FC
- INPUT $->$ [CONV $->$ RELU $->$ POOL]*2 $->$ FC $->$ RELU $->$ FC. Here we see that there is a single CONV layer between every POOL layer.
- INPUT $->$ [CONV $->$ RELU $->$ CONV $->$ RELU $->$ POOL]*3 $->$ [FC $->$ RELU]*2 $->$ FC Here we see two CONV layers stacked before every POOL layer.
- This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

ConvNet Architectures

- Prefer a stack of small filter CONV to one large receptive field CONV layer.

ConvNet Architectures

- Prefer a stack of small filter CONV to one large receptive field CONV layer.
- Intuitively, stacking CONV layers with tiny filters as opposed to having one CONV layer with big filters allows us to express more powerful features of the input, and with fewer parameters.

ConvNet Architectures

- Prefer a stack of small filter CONV to one large receptive field CONV layer.
- Intuitively, stacking CONV layers with tiny filters as opposed to having one CONV layer with big filters allows us to express more powerful features of the input, and with fewer parameters.
- As a practical disadvantage, we might need more memory to hold all the intermediate CONV layer results if we plan to do backpropagation.

ImageNet

- In practice: use whatever works best on ImageNet.

ImageNet

- In practice: use whatever works best on ImageNet.
- Instead of rolling your own architecture for a problem, you should look at whatever architecture currently works best on ImageNet, download a pretrained model and finetune it on your data.

Layer Sizing Patterns

- The input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.

Layer Sizing Patterns

- The input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.
- The conv layers should be using small filters (e.g. 3×3 or at most 5×5), using a stride of $S = 1$, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input.

Layer Sizing Patterns

- The input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.
- The conv layers should be using small filters (e.g. 3×3 or at most 5×5), using a stride of $S = 1$, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input.
- That is, when $F = 3$, then using $P = 1$ will retain the original size of the input.

Layer Sizing Patterns

- The input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.
- The conv layers should be using small filters (e.g. 3×3 or at most 5×5), using a stride of $S = 1$, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input.
- That is, when $F = 3$, then using $P = 1$ will retain the original size of the input.
- When $F = 5, P = 2$.

Layer Sizing Patterns

- The input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.
- The conv layers should be using small filters (e.g. 3×3 or at most 5×5), using a stride of $S = 1$, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input.
- That is, when $F = 3$, then using $P = 1$ will retain the original size of the input.
- When $F = 5, P = 2$.
- For a general F, it can be seen that $P = (F - 1)/2$ preserves the input size.

Layer Sizing Patterns

- If you must use bigger filter sizes (such as 7×7 or so), it is only common to see this on the very first Conv layer that is looking at the input image.

Layer Sizing Patterns

- If you must use bigger filter sizes (such as 7×7 or so), it is only common to see this on the very first Conv layer that is looking at the input image.
- The pool layers are in charge of downsampling the spatial dimensions of the input. The most common setting is to use max-pooling with 2x2 receptive fields (i.e. $F = 2$), and with a stride of 2 (i.e. $S = 2$).

Layer Sizing Patterns

- If you must use bigger filter sizes (such as 7×7 or so), it is only common to see this on the very first Conv layer that is looking at the input image.
- The pool layers are in charge of downsampling the spatial dimensions of the input. The most common setting is to use max-pooling with 2x2 receptive fields (i.e. $F = 2$), and with a stride of 2 (i.e. $S = 2$).
- Note that this discards exactly 75% of the activations in an input volume (due to downsampling by 2 in both width and height).

Reducing sizing headaches

- The scheme presented before is pleasing because all the CONV layers preserve the spatial size of their input, while the POOL layers alone are in charge of down-sampling the volumes spatially.

Reducing sizing headaches

- The scheme presented before is pleasing because all the CONV layers preserve the spatial size of their input, while the POOL layers alone are in charge of down-sampling the volumes spatially.
- In an alternative scheme where we use strides greater than 1 or don?t zero-pad the input in CONV layers, we would have to very carefully keep track of the input volumes throughout the CNN architecture and make sure that all strides and filters work out, and that the ConvNet architecture is nicely and symmetrically wired.

Why use stride of 1 in CONV?

- Smaller strides work better in practice.

Why use stride of 1 in CONV?

- Smaller strides work better in practice.
- Additionally, stride 1 allows us to leave all spatial down-sampling to the POOL layers, with the CONV layers only transforming the input volume depth-wise.

Why use padding?

- In addition to the aforementioned benefit of keeping the spatial sizes constant after CONV, doing this actually improves performance.

Why use padding?

- In addition to the aforementioned benefit of keeping the spatial sizes constant after CONV, doing this actually improves performance.
- If the CONV layers were to not zero-pad the inputs and only perform valid convolutions, then the size of the volumes would reduce by a small amount after each CONV, and the information at the borders would be washed away too quickly.

Compromising based on memory constraints

- In some cases, the amount of memory can build up very quickly with the rules of thumb presented before.

Compromising based on memory constraints

- In some cases, the amount of memory can build up very quickly with the rules of thumb presented before.
- For example, filtering a $224 \times 224 \times 3$ image with three 3×3 CONV layers with 64 filters each and padding 1 would create three activation volumes of size $[224 \times 224 \times 64]$.

Compromising based on memory constraints

- In some cases, the amount of memory can build up very quickly with the rules of thumb presented before.
- For example, filtering a $224 \times 224 \times 3$ image with three 3×3 CONV layers with 64 filters each and padding 1 would create three activation volumes of size $[224 \times 224 \times 64]$.
- This amounts to a total of about 10 million activations, or 72MB of memory (per image, for both activations and gradients).

Compromising based on memory constraints

- In some cases, the amount of memory can build up very quickly with the rules of thumb presented before.
- For example, filtering a $224 \times 224 \times 3$ image with three 3×3 CONV layers with 64 filters each and padding 1 would create three activation volumes of size $[224 \times 224 \times 64]$.
- This amounts to a total of about 10 million activations, or 72MB of memory (per image, for both activations and gradients).
- Since GPUs are often bottlenecked by memory, it may be necessary to compromise.

Compromising based on memory constraints

- In practice, people prefer to make the compromise at only the first CONV layer of the network.

Compromising based on memory constraints

- In practice, people prefer to make the compromise at only the first CONV layer of the network.
- For example, one compromise might be to use a first CONV layer with filter sizes of 7×7 and stride of 2 (as seen in a ZF net).

Compromising based on memory constraints

- In practice, people prefer to make the compromise at only the first CONV layer of the network.
- For example, one compromise might be to use a first CONV layer with filter sizes of 7×7 and stride of 2 (as seen in a ZF net).
- As another example, an AlexNet uses filter sizes of 11×11 and stride of 4.

Case Studies

- LeNet: The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990?s. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.

Case Studies

- LeNet: The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990?s. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.
- AlexNet: The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error).

Case Studies

- LeNet: The first successful applications of Convolutional Networks were developed by Yann LeCun in 1990?s. Of these, the best known is the LeNet architecture that was used to read zip codes, digits, etc.
- AlexNet: The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error).
- The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other

Case Studies

- ZF Net: The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

Case Studies

- ZF Net: The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.
- GoogLeNet: The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).

Case Studies

- VGGNet: The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman. Its main contribution was in showing that the depth of the network is a critical component for good performance.

Case Studies

- VGGNet: The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman. Its main contribution was in showing that the depth of the network is a critical component for good performance.
- Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.

Case Studies

- VGGNet: The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman. Its main contribution was in showing that the depth of the network is a critical component for good performance.
- Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- ResNet: Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015.

Case Studies

- VGGNet: The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman. Its main contribution was in showing that the depth of the network is a critical component for good performance.
- Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.
- ResNet: Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015.
- ResNets are currently by far state of the art Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 10, 2016).