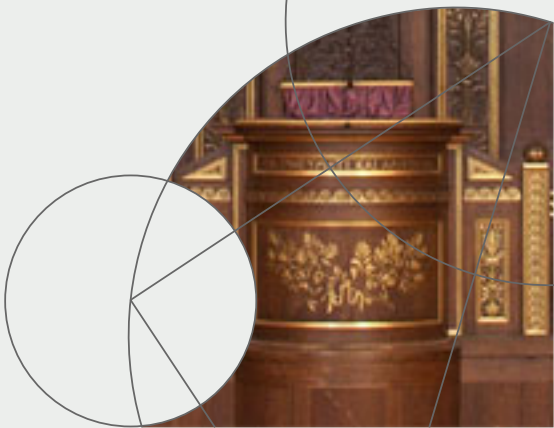
Deep Learning: Convolutional Neural Network

Rensheng Wang,
https://sit.instructure.com/courses/22680
August 31, 2019



Convolutional Neural Network

- Convolutional neural networks (CNNs) emerged from the study of the brains visual cortex, and they have been used in image recognition since the 1980s.
- Recently, thanks to the increase in computational power, the amount of available training data, and the tricks for training deep nets, CNNs have managed to achieve superhuman performance on some complex visual tasks.
- They power image search services, self-driving cars, automatic video classification systems, and more. Moreover, CNNs are not restricted to visual perception: they are also successful at other tasks, such as voice recognition or natural language processing (NLP);
- Here we will focus on visual applications for now.

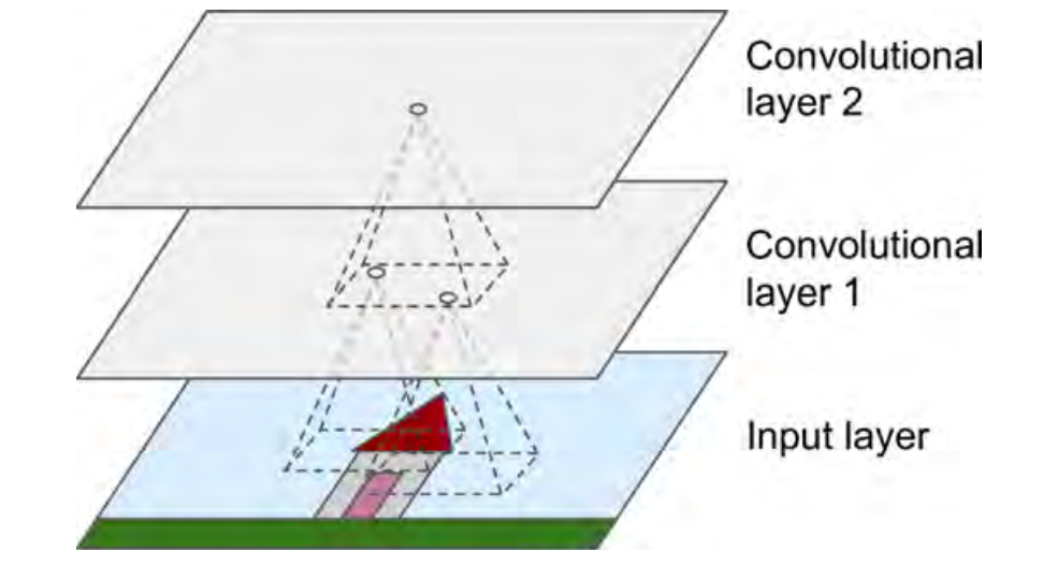
Convolutional Neural Network

banking (#s on checks when you deposit in ATM)

- An important milestone was a 1998 paper by Yann LeCun, et al, which introduced the famous LeNet-5 architecture, widely used to recognize handwritten check numbers.
- This architecture introduces two new building blocks: convolutional layers and pooling layers.
- The most important building block of a CNN is the convolutional layer: neurons in the first convolutional layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields.
- In turn, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer.
- This architecture allows the network to concentrate on low-level features in the first hidden layer, then assemble them into higher-level features in the next hidden layer, and so on.
- This hierarchical structure is common in real-world images, which is one of the reasons why CNNs work so well for image recognition.

Convolutional Layer

- CNN layers with rectangular local receptive fields



TensorFlow Implementation

- In TensorFlow, each input image is typically represented as a 3D tensor of $\texttt{shape}[\texttt{height}, \texttt{width}, \texttt{channels}]$.
- A mini-batch is represented as a 4D tensor of $\texttt{shape}[\texttt{mini} - \texttt{batchsize}, \texttt{height}, \texttt{width}, \texttt{channels}]$.
- The weights of a convolutional layer are represented as a 4D tensor of $\texttt{shape}[\texttt{f}_\texttt{h}, \texttt{f}_\texttt{w}, \texttt{f}_\texttt{n}, \texttt{f}'_\texttt{n}]$.
- The bias terms of a convolutional layer are simply represented as a 1D tensor of $\texttt{shape}[\texttt{f}_\texttt{n}]$.

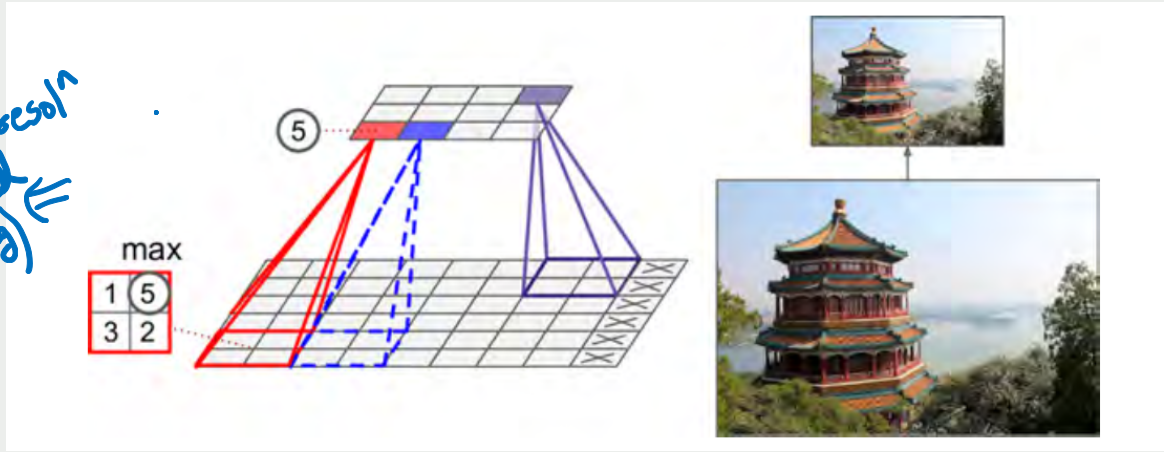
Pooling Layer

↳ lowering the resolⁿ

- The goal for pooling layer is to subsample (i.e., shrink) the input image in order to reduce the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting).
- Reducing the input image size also makes the neural network tolerate a little bit of image shift (location invariance).
- Just like in convolutional layers, each neuron in a pooling layer is connected to the outputs of a limited number of neurons in the previous layer, located within a small rectangular receptive field. You must define its size, the stride, and the padding type, just like before.
- However, a pooling neuron has no weights; all it does is aggregate the inputs using an aggregation function such as the max or mean.

Max Pooling Layer

- Max pooling layer is the most common type of pooling layer. Note that only the max input value in each kernel makes it to the next layer. The other inputs are dropped
- Max pooling layer (2 X 2 pooling kernel, stride 2, no padding)



LeNet-5

- The LeNet-5 architecture is perhaps the most widely known CNN architecture. It was created by Yann LeCun in 1998 and widely used for hand written digit recognition (MNIST).
- It is composed of the layers shown below.

84 was an arbitrary choice

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

AlexNet

- It was developed by Alex Krizhevsky et al (hence the name).
- It is quite similar to LeNet-5, only much larger and deeper, and it was the first to stack convolutional layers directly on top of each other, instead of stacking a pooling layer on top of each convolutional layer

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13×13	3×3	1	SAME	ReLU
C6	Convolution	384	13×13	3×3	1	SAME	ReLU
C5	Convolution	384	13×13	3×3	1	SAME	ReLU
S4	Max Pooling	256	13×13	3×3	2	VALID	–
C3	Convolution	256	27×27	5×5	1	SAME	ReLU
S2	Max Pooling	96	27×27	3×3	2	VALID	–
C1	Convolution	96	55×55	11×11	4	SAME	ReLU
In	Input	3 (RGB)	224×224	–	–	–	–

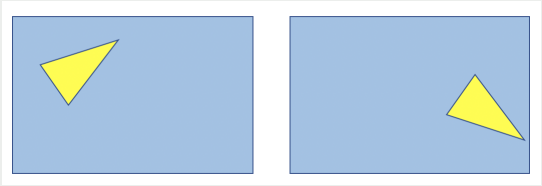
Convolutionary Neural Network (CNN)

- Connecting the neurons in a special way to improve image processing
- Why CNN in Image?
  - The image size can be very large. The fully connected network can be simplified by utilizing some properties in image recognition.
  - Some patterns are much smaller than the whole image (a neuron does not have to see the whole image to discover the pattern)

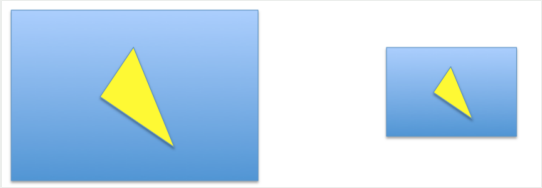
conv on local instead of global

Convolutionary Neural Network (CNN)

- Why CNN in Image?
  - The same pattern appears in different regions

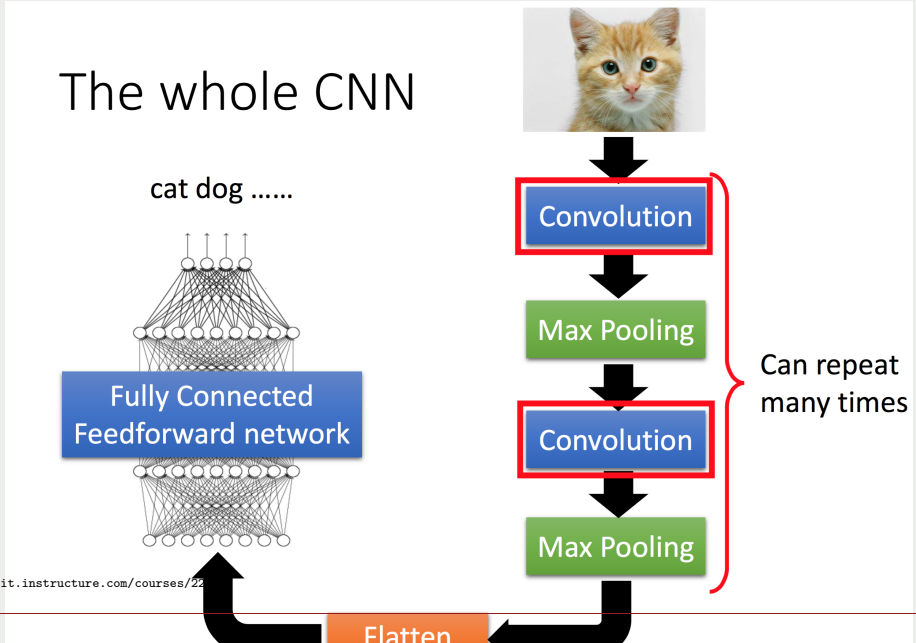


  - Subsampling the pixels will not change the object (lower resolution with less parameters)



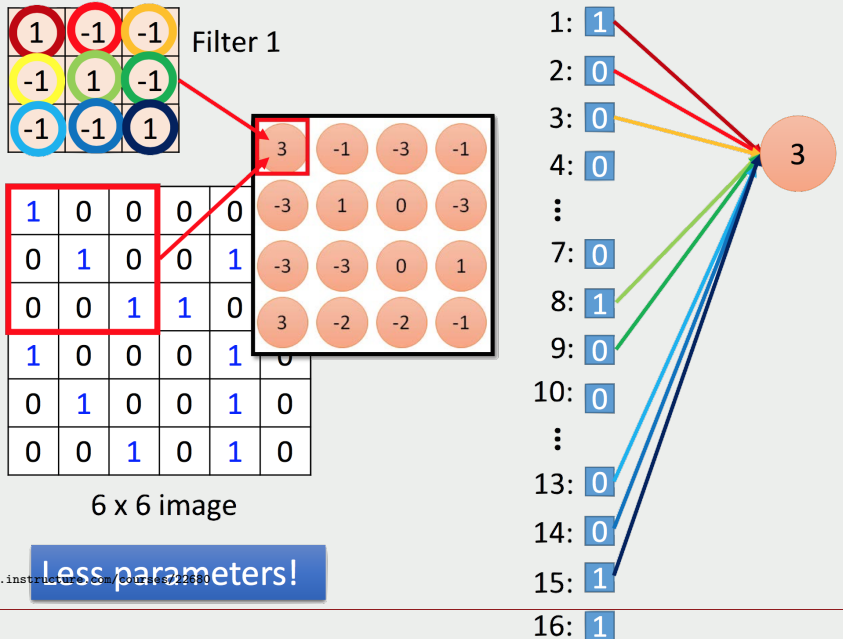
Convolutionary Neural Network (CNN)

- Typical procedures for CNN



CNN-Convolutional Filter

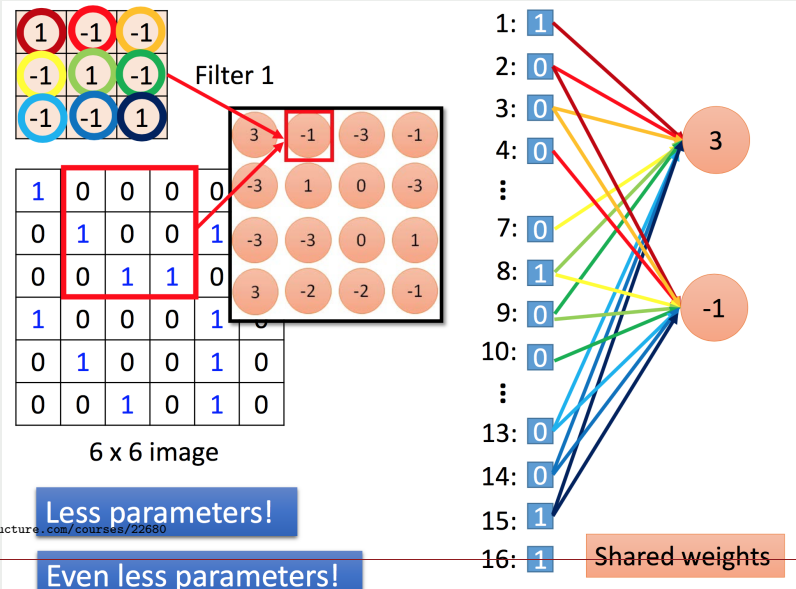
- For below 6×6 image with 36 pixels, only 9 connected instead of fully connected



Less parameters!

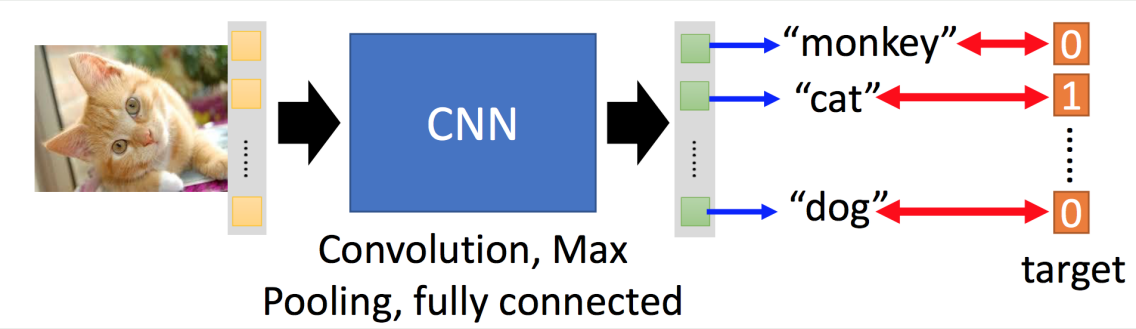
CNN-Convolutional Filter

- For below 6×6 image with 36 pixels, only 9 connected instead of fully connected
- All neurons share the same 9 weight parameters (even less parameters)



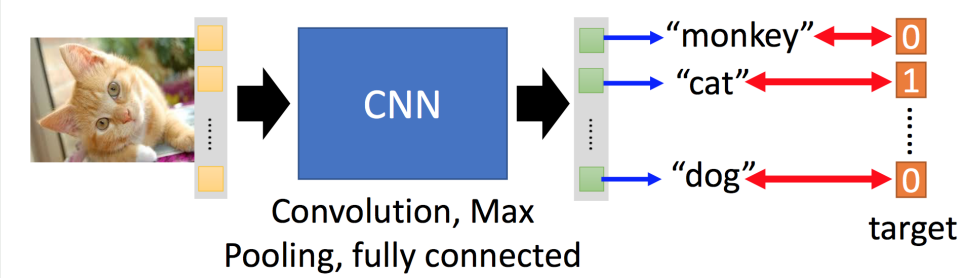
CNN-Learning

- For CNN, like before, use gradient descent to learn the target



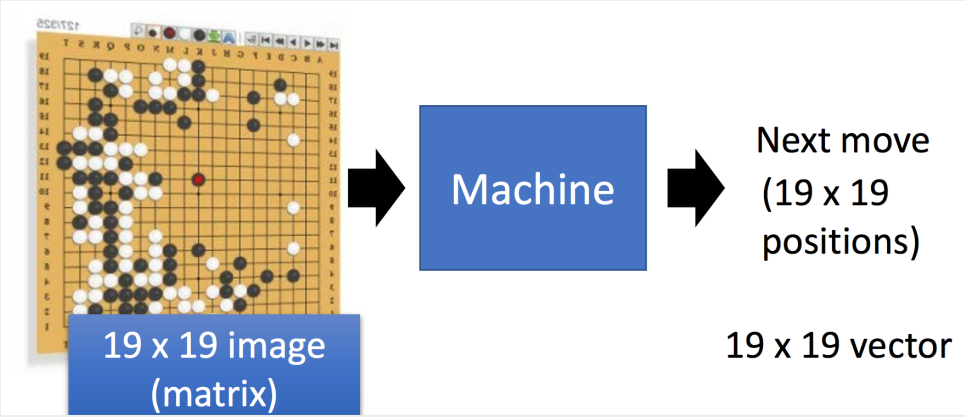
CNN-Learning

- For CNN, like before, use gradient descent to learn the target
- Here we ignored the non-linear activation function after the convolution to simplify the demonstrations .
- We need to tune the NN structure to fit the model



CNN-Playing Go

- Denote Black : 1; White : -1; Empty : 0;

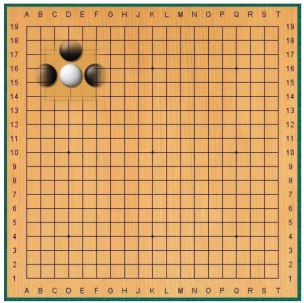


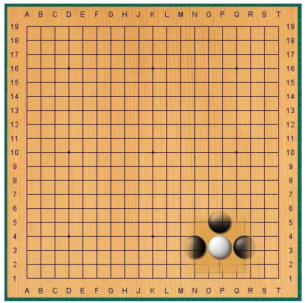
Why CNN for Playing Go

- CNN Property 1: Some pattern much smaller than the whole image

  Alpha Go use 5×5 for first layer filter

- CNN Property 2: The same patterns appear in different regions



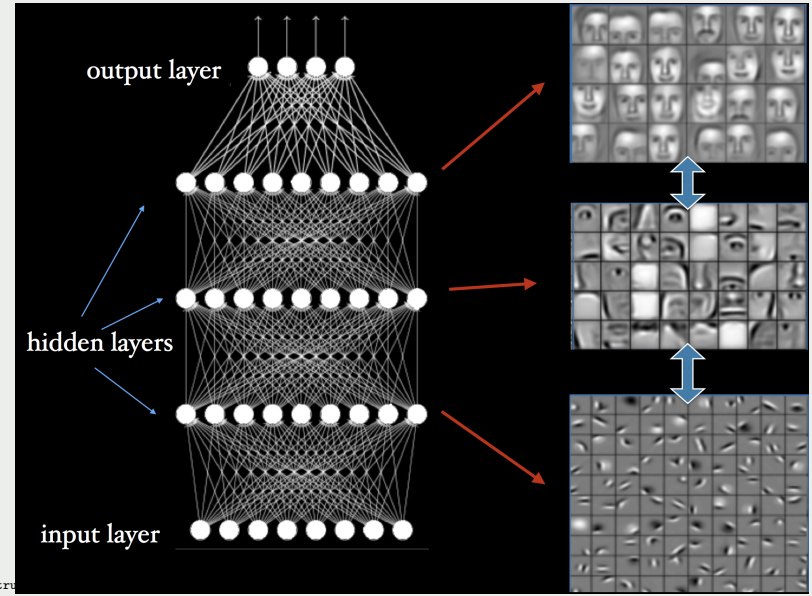


Why CNN for Playing Go

- CNN Property 3: Subsampling will not change the pattern (Max-Pooling)
  - Alpha Go use 5×5 for first layer filter, in total 48 filters
  - Alpha Go use zero-padding to keep the same size after the filtering
  - Alpha Go does not use Max-pooling!

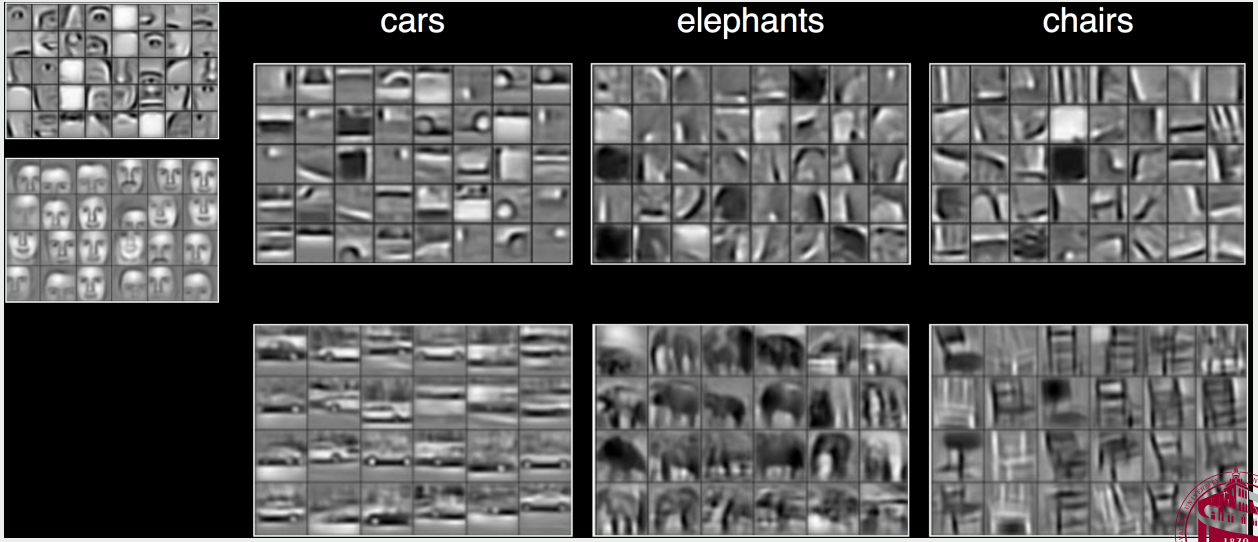
CNN: Feature hierarchies

- Man-face recognition



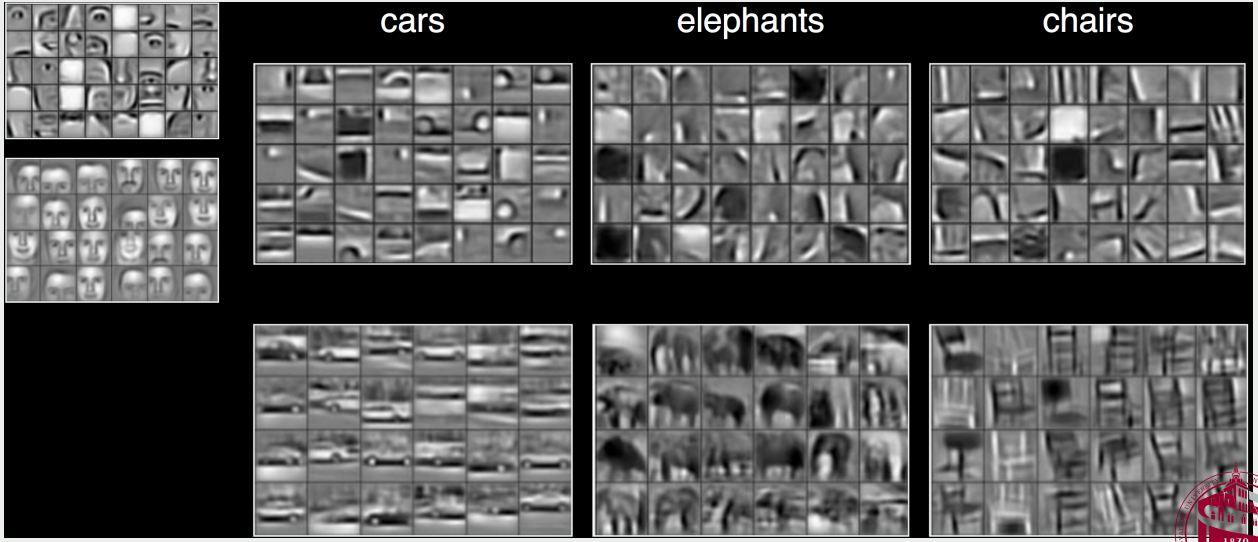
CNN: Feature hierarchies

- More images: cars, elephants, chairs



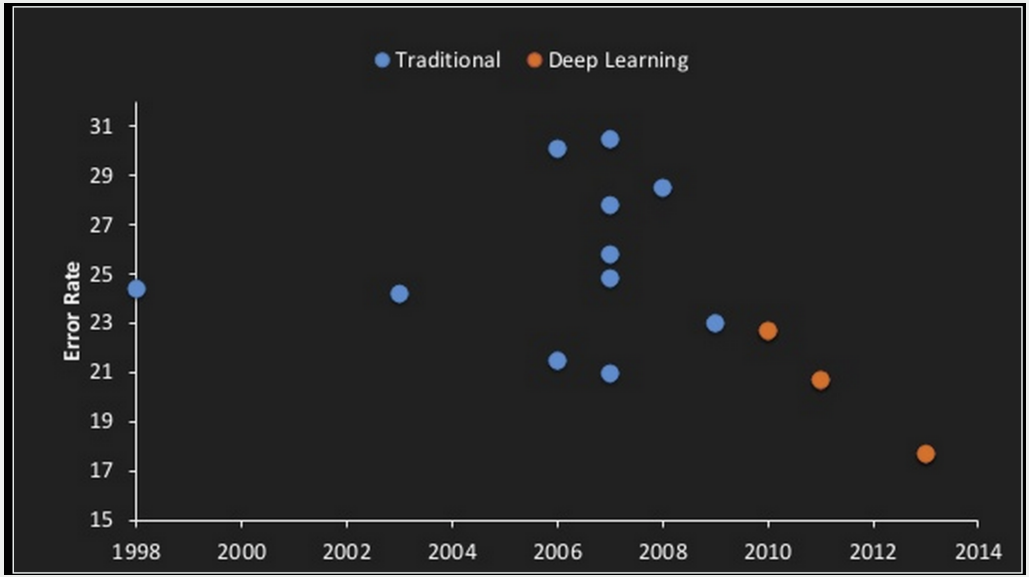
CNN: Feature hierarchies

- More images: cars, elephants, chairs



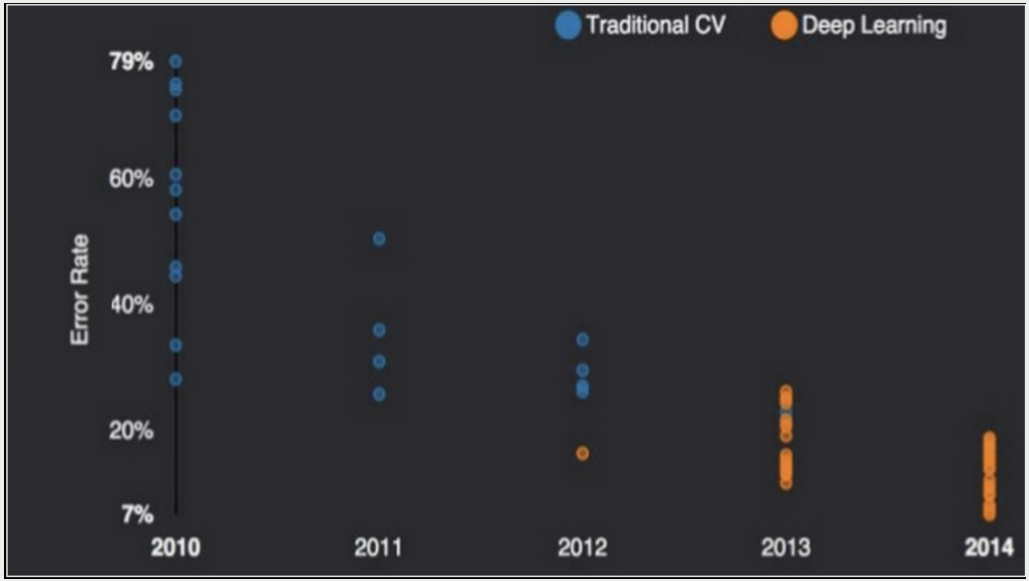
CNN: Audio Recognition

- CNN vs. Traditional



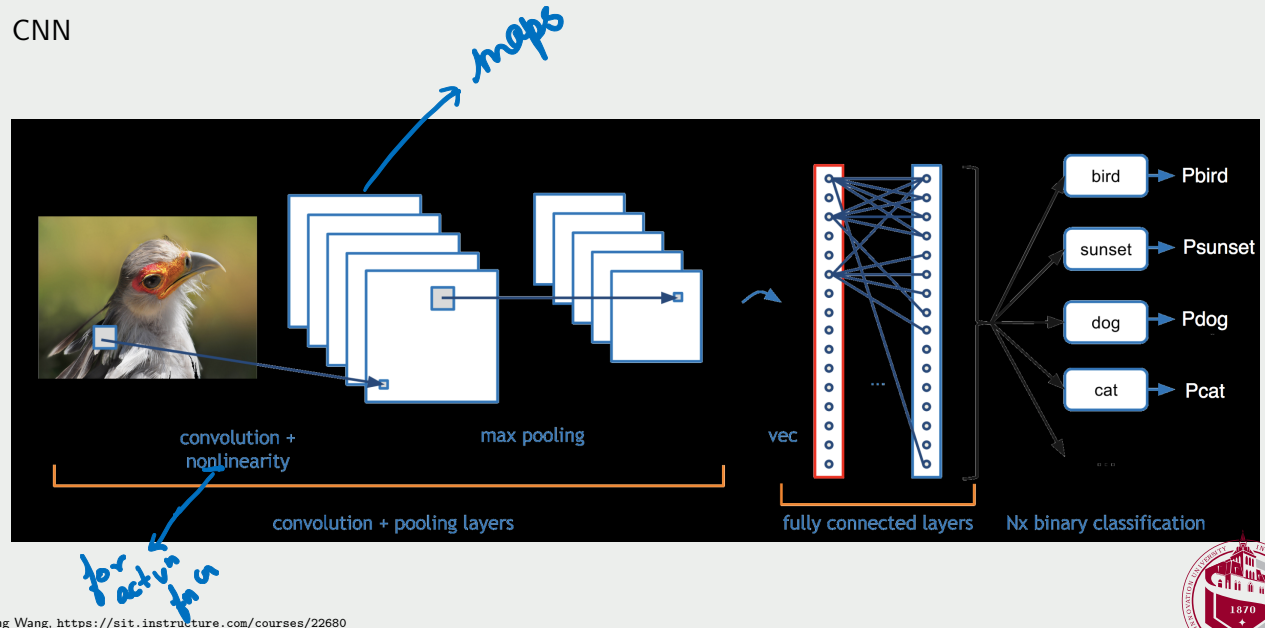
CNN: Image Recognition

- CNN vs. Traditional



CNN: Processing Flow Chart

- CNN

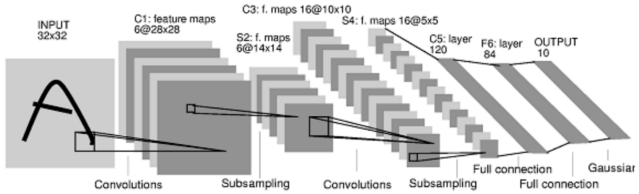


CNN: History

- CNN

1998

LeCun et al.



# of transistors

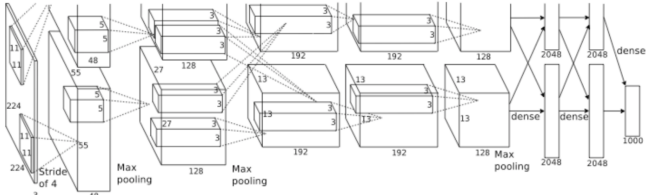
10^6

# of pixels used in training

10^7 NIST

2012

Krizhevsky et al.



# of transistors

10^9

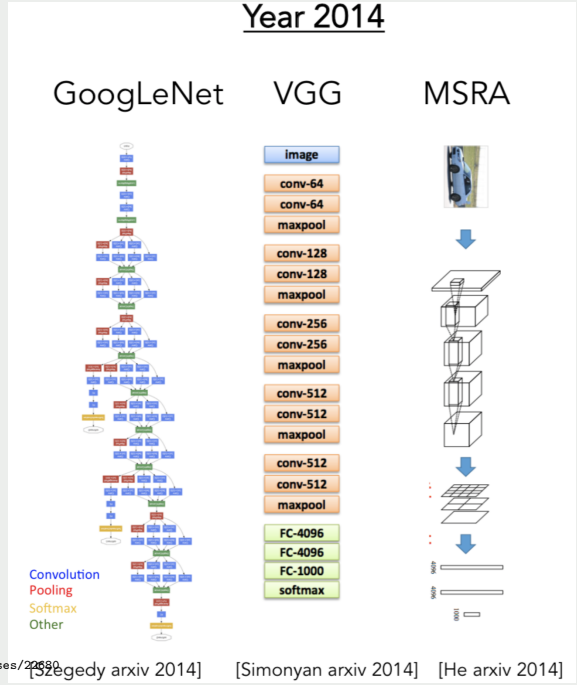
GPUs

# of pixels used in training

10^{14} IMAGENET

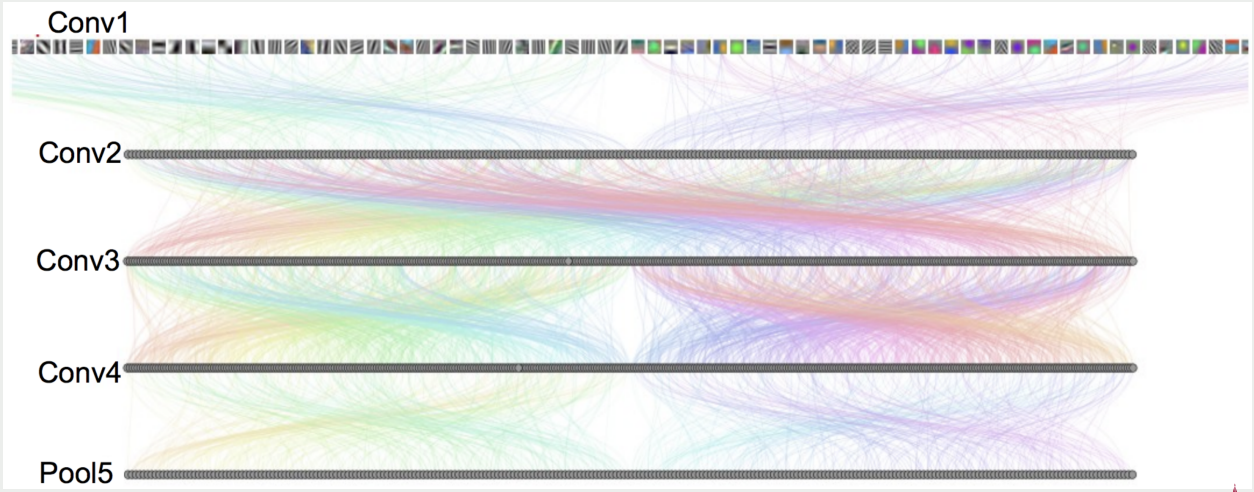
CNN: History

- CNN



CNN: Visualization

- CNN



CNN: Training

- CNN

```
net = NeuralNet(
    layers=[
        ('input', layers.InputLayer),
        ('conv1', Conv2DLayer),
        ('pool1', MaxPool2DLayer),
        ('dropout1', layers.DropoutLayer),
        ('conv2', Conv2DLayer),
        ('pool2', MaxPool2DLayer),
        ('dropout2', layers.DropoutLayer),
        ('conv3', Conv2DLayer),
        ('pool3', MaxPool2DLayer),
        ('dropout3', layers.DropoutLayer),
        ('hidden4', layers.DenseLayer),
        ('dropout4', layers.DropoutLayer),
        ('hidden5', layers.DenseLayer),
        ('output', layers.DenseLayer),
        ],
    input_shape=(None, 1, 96, 96),
    conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_pool_size=(2, 2),
    dropout1_p=0.1,
    conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_pool_size=(2, 2),
    dropout2_p=0.2,
    conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_pool_size=(2, 2),
    dropout3_p=0.3,
    hidden4_num_units=1000,
    dropout4_p=0.5,
    hidden5_num_units=1000,
    output_num_units=30, output_nonlinearity=None,

    update_learning_rate=theano.shared(float32(0.03)),
    update_momentum=theano.shared(float32(0.9)),
```

layer definitions

layer parameters

CNN: Training

```
net = NeuralNet(
    layers=[
        ('input', layers.InputLayer),
        ('conv1', Conv2DLayer),
        ('pool1', MaxPool2DLayer),
        ('dropout1', layers.DropoutLayer),
        ('conv2', Conv2DLayer),
        ('pool2', MaxPool2DLayer),
        ('dropout2', layers.DropoutLayer),
        ('conv3', Conv2DLayer),
        ('pool3', MaxPool2DLayer),
        ('dropout3', layers.DropoutLayer),
        ('hidden4', layers.DenseLayer),
        ('dropout4', layers.DropoutLayer),
        ('hidden5', layers.DenseLayer),
        ('output', layers.DenseLayer),
        ],
    input_shape=(None, 1, 96, 96),
    conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_pool_size=(2, 2),
    dropout1_p=0.1,
    conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_pool_size=(2, 2),
    dropout2_p=0.2,
    conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_pool_size=(2, 2),
    dropout3_p=0.3,
    hidden4_num_units=1000,
    dropout4_p=0.5,
    hidden5_num_units=1000,
    output_num_units=30, output_nonlinearity=None,

    update_learning_rate=theano.shared(float32(0.03)),
    update_momentum=theano.shared(float32(0.9)),

    batch_iterator_train=FlipBatchIterator(batch_size=48),
    on_epoch_finished=[
        AdjustVariable('update_learning_rate', start=0.02, stop=0.00001),
        AdjustVariable('update_momentum', start=0.9, stop=0.999),
        EarlyStopping(patience=250),
        ],
    max_epochs=10000,
    verbose=1,
    )
```

hyper parameters

CNN: Training

- CNN

Lift off!

```
X, y = loadstuff()
net = NeuralNet(...)
net.fit(X, y)
with open('/mnt/nets/netx.pickle', 'wb') as f:
    pickle.dump(net, f, -1)
```

epoch	train loss	valid loss	train/val	dur
1	0.11094	0.04377	2.53447	2.97s
2	0.01819	0.00842	2.16100	2.98s
3	0.00800	0.00707	1.13217	2.98s
4	0.00671	0.00667	1.00530	2.97s
5	0.00635	0.00631	1.00533	2.97s
6	0.00611	0.00608	1.00492	2.98s
7	0.00592	0.00587	1.00827	2.97s
8	0.00575	0.00569	1.01183	2.97s
9	0.00561	0.00552	1.01514	2.97s
10	0.00548	0.00538	1.01815	2.97s

CNN: Training

- Visualize Loss Curve

```
import numpy as np
import matplotlib.pyplot as pyplot
%matplotlib inline

import cPickle as pickle
with open('/mnt/nets/netx.pickle', 'rb') as f:
    net = pickle.load(f)

train_loss = np.array([i["train_loss"] for i in net.train_history_])
valid_loss = np.array([i["valid_loss"] for i in net.train_history_])
pyplot.plot(train_loss, linewidth=3, label="train")
pyplot.plot(valid_loss, linewidth=3, label="valid")
pyplot.grid()
pyplot.legend()
pyplot.xlabel("epoch")
pyplot.ylabel("loss")
pyplot.ylim(1e-4*5, 1e-2)
pyplot.yscale("log")
pyplot.show()
```

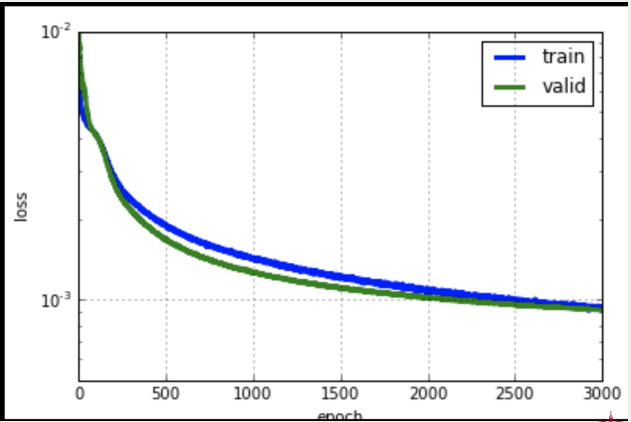
CNN: Training

- Visualize Loss Curve

```
import numpy as np
import matplotlib.pyplot as pyplot
%matplotlib inline

import cPickle as pickle
with open('/mnt/nets/netx.pickle', 'rb') as f:
    net = pickle.load(f)

train_loss = np.array([i["train_loss"] for i in net.train_history_])
valid_loss = np.array([i["valid_loss"] for i in net.train_history_])
pyplot.plot(train_loss, linewidth=3, label="train")
pyplot.plot(valid_loss, linewidth=3, label="valid")
pyplot.grid()
pyplot.legend()
pyplot.xlabel("epoch")
pyplot.ylabel("loss")
pyplot.ylim(1e-4*5, 1e-2)
pyplot.yscale("log")
pyplot.show()
```



CNN: Training

- Visualize Loss Curve