EE269
Signal Processing for Machine Learning

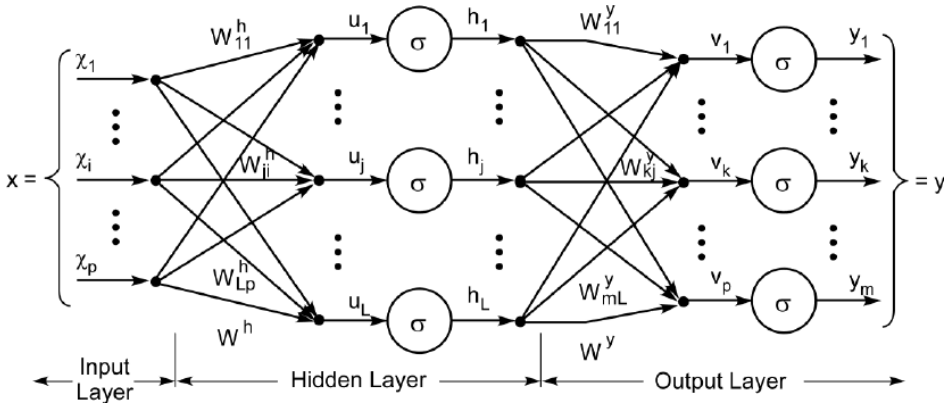
Lecture 16

Instructor : Mert Pilanci

Stanford University

November 18, 2020

Recap: Multilayer Neural Networks



$$z^{(0)} = x \quad \text{(input)}$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \qquad l = 1, ..., L$$

$$z_j^{(l)} = \sigma(a_k^l) \qquad l = 1, ..., L$$

- Training: Parameters $\Theta = (W^{(1)}, W^{(2)}, ..., W^{(L)})$
  regression: (squared loss) vs classification (cross-entropy loss)

$$\min_\Theta \sum_{n=1}^{N} (y_n - f(x_n))^2 \qquad \min_\Theta - \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \log f_k(x_n)$$

Training Multilayer Neural Networks

- Training: Parameters $\Theta = (W^{(1)}, W^{(2)}, ..., W^{(L)})$

  regression: (squared loss) vs classification (cross-entropy loss)

$$\min_{\Theta} \sum_{n=1}^{N} \underbrace{(y_n - f(x_n))^2}_{R_n(\Theta)} \qquad \min_{\Theta} - \sum_{n=1}^{N} \underbrace{\sum_{k=1}^{K} y_{nk} \log f_k(x_n)}_{R_n(\Theta)}$$

- Gradient Descent

  $\Theta_{t+1} = \Theta_{t+1} - \sum_{i=1}^{n} \frac{\partial}{\partial \Theta} R_n(\Theta)$
- Stochastic Gradient Descent

  $\Theta_{t+1} = \Theta_{t+1} - \frac{\partial}{\partial \Theta} R_{n_t}(\Theta)$

  where n_t is a random index
- non-convex optimization problem

Computing derivatives: Backpropagation Algorithm

$$z^{(0)} = x \quad \text{(input)}$$

$$a_j^{(l)} = \sum_i W_{ij}^{(l)} z_i^{(l-1)} \qquad l = 1, ..., L$$

$$z_j^{(l)} = \sigma(a_k^l) \qquad l = 1, ..., L$$

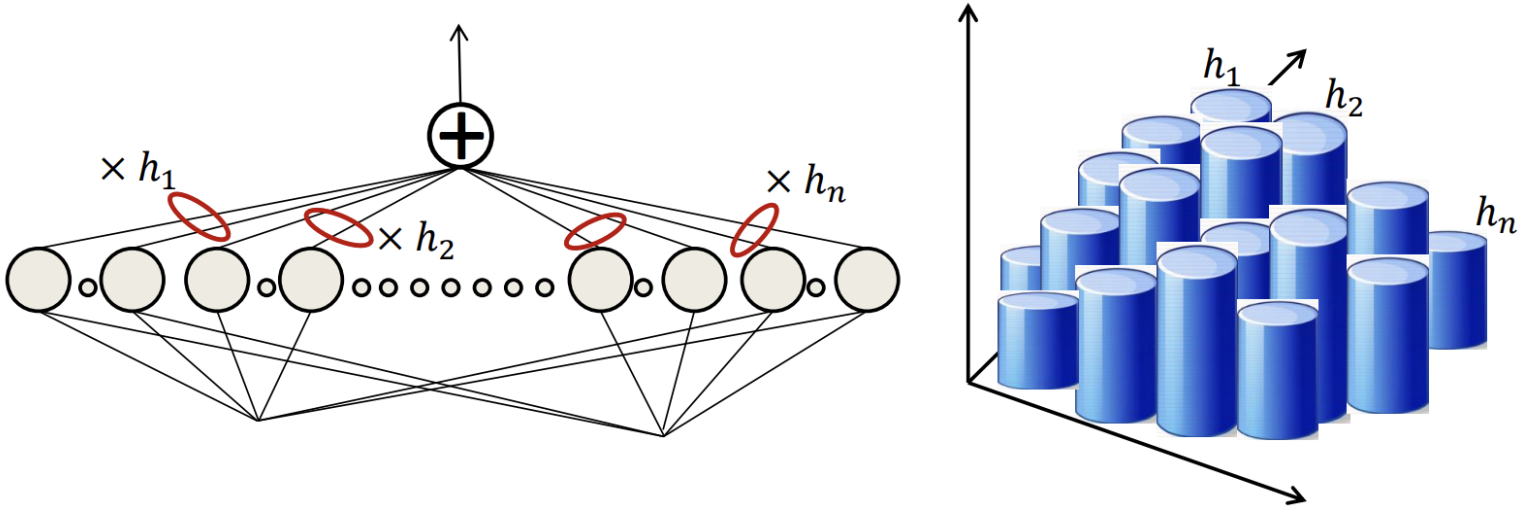
$$\min_{\Theta} \sum_{n=1}^{N} \underbrace{(y_n - f(x_n))^2}_{R_n(\Theta)}$$

$$\frac{\partial R_n(\Theta)}{\partial W_{ij}^{(l)}} = \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial W_{ij}^{(l)}} = \delta_{nj}^{(l)} z_i^{(l-1)}$$

where we defined

$$\delta_{nj}^{(l)} \triangleq \frac{\partial R_n(\Theta)}{\partial a_j^{(l)}}$$

Two layer networks are universal approximators



- Two layer networks (also called one hidden layer) can compose arbitrary functions to arbitrary precision
- May require infinitely many neurons in the hidden layer
- Deeper networks require **fewer** neurons for the same approximation error

Overparametrization

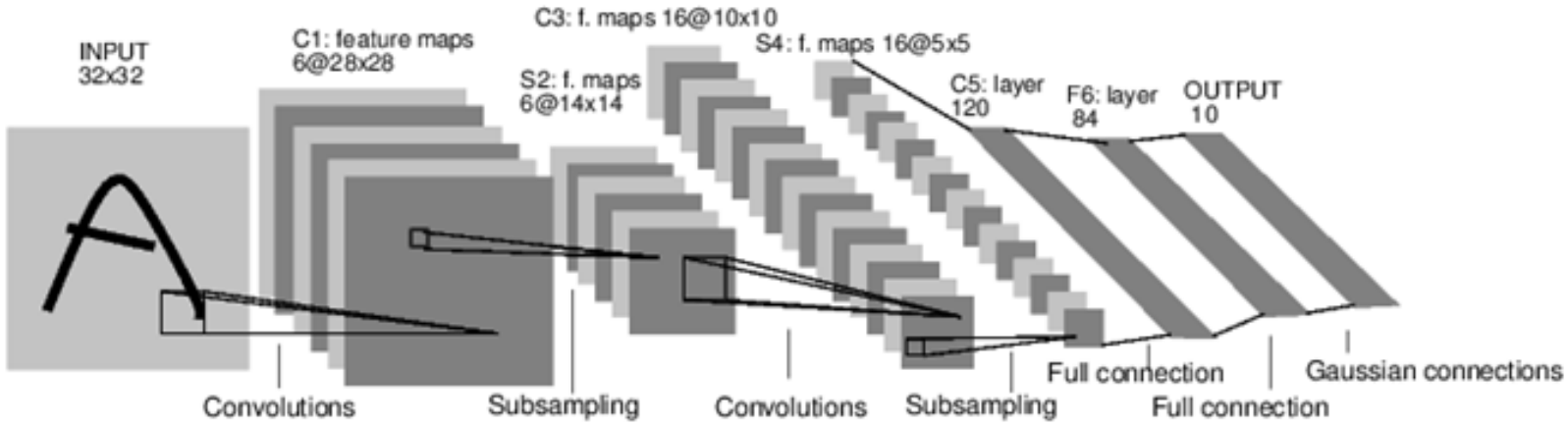
- $(u)_+ := \max(0, u)$ (ReLU activation) denotes the positive part of a scalar
- Let $y \in \pm 1$ denote training labels

  Two layer scalar output ReLU network:

$$f(x) = \sum_{j=1}^{m} w_j^{(2)} (x^T w_j^{(1)})_+$$

- The equation $f(x) = y$ (zero training loss) can have multiple solutions
- The numerical optimizer (gradient descent, stochastic gradient, momentum etc) may have an inductive bias

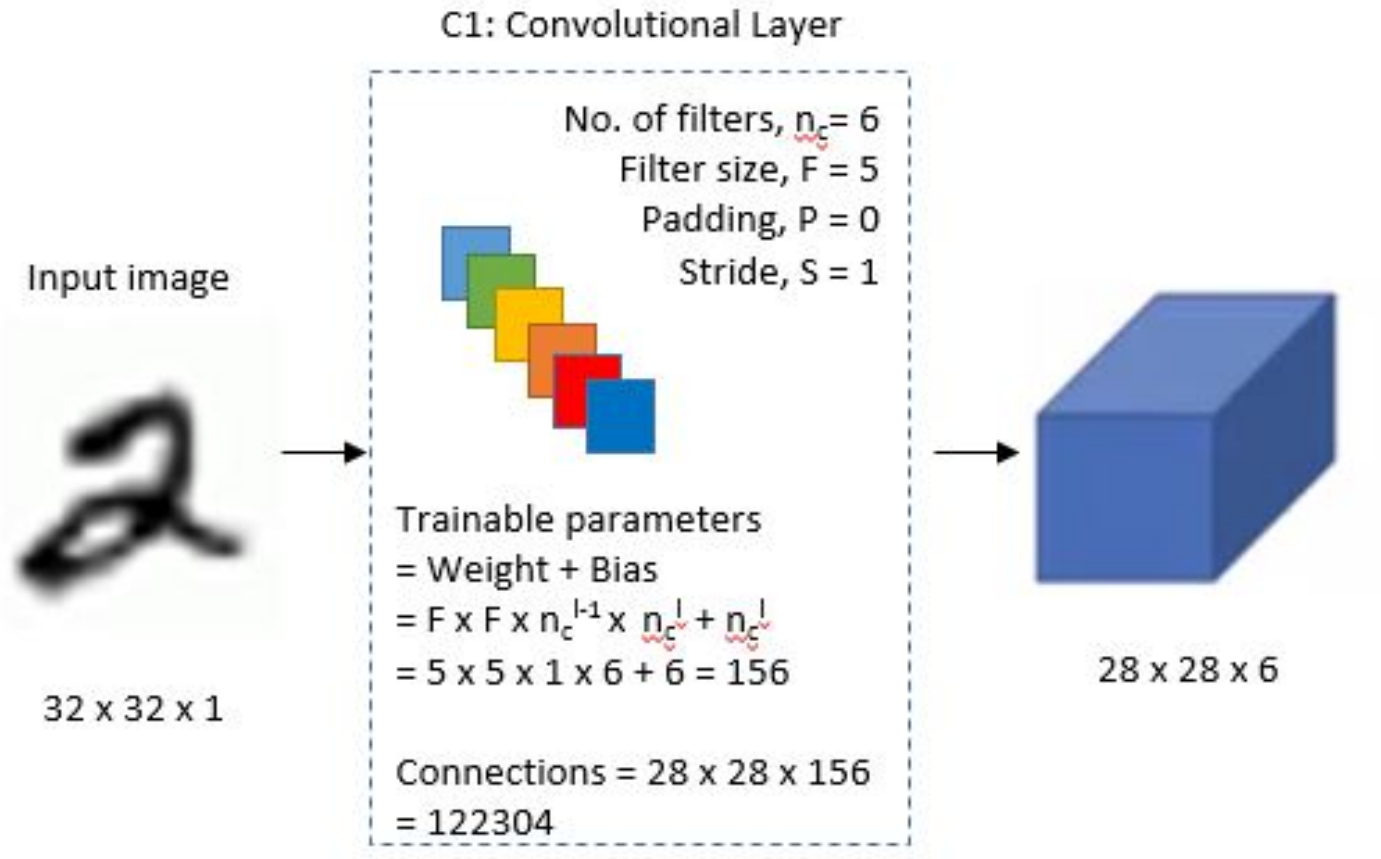
Convolutional Neural Networks (CNNs)



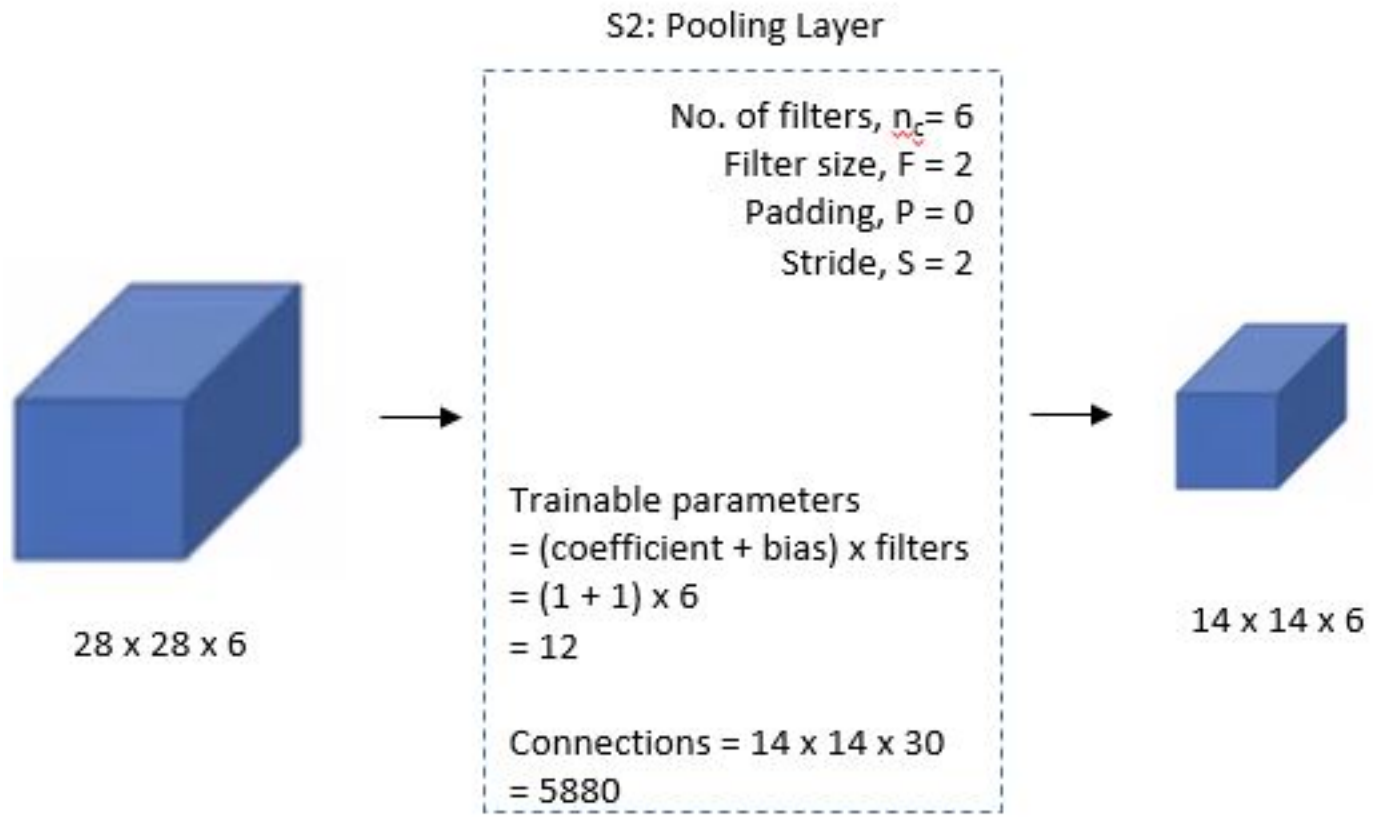
A Full Convolutional Neural Network (LeNet)

Gradient-based learning applied to document recognition. LeCun et al., 1998

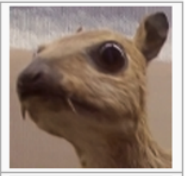
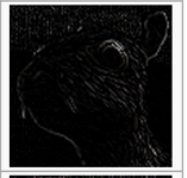
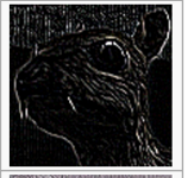
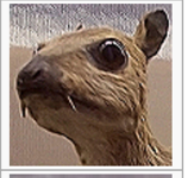
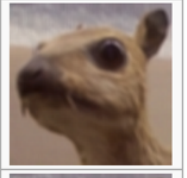
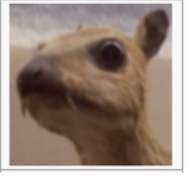
Convolutional Neural Networks (CNNs)



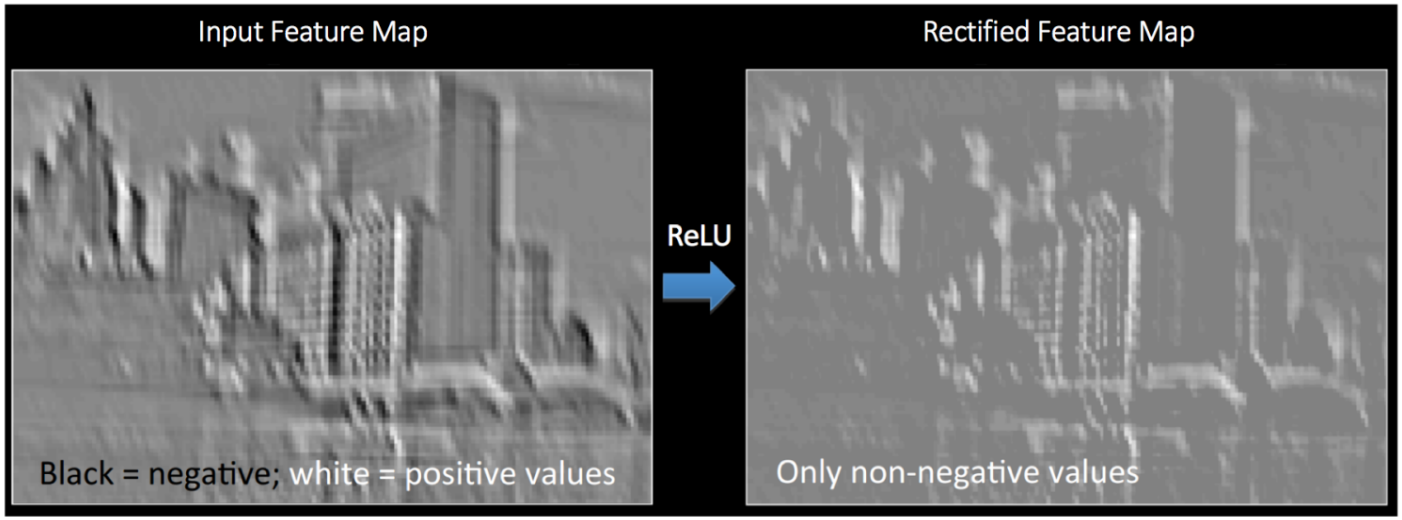
Convolutional Neural Networks (CNNs)



2D Convolution

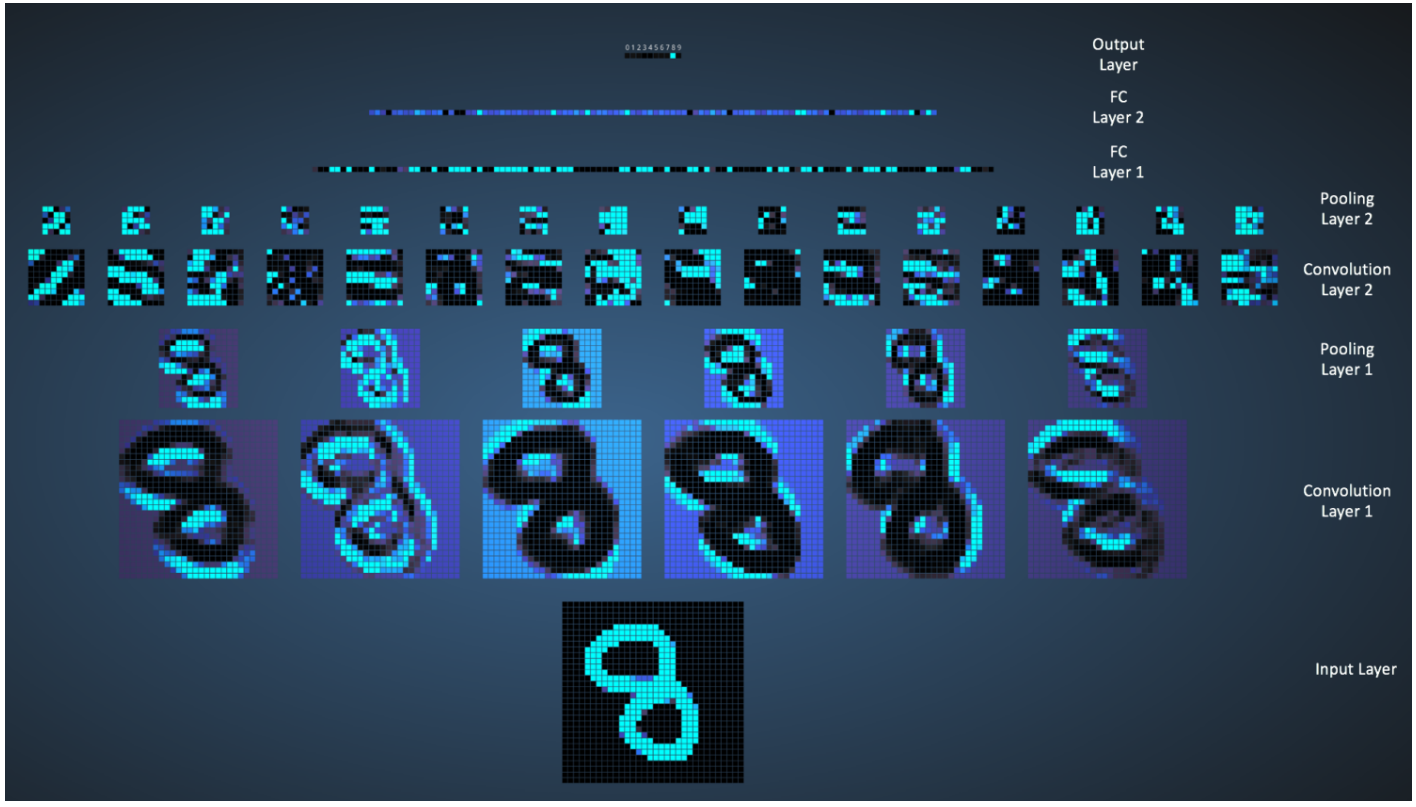
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Nonlinearity: Rectified Linear Unit



slide credit: R. Fergus

MNIST digit classification using LeNet



slide credit: A. Harley. http://scs.ryerson.ca/ aharley/vis/conv/flat.html

Imagenet Dataset

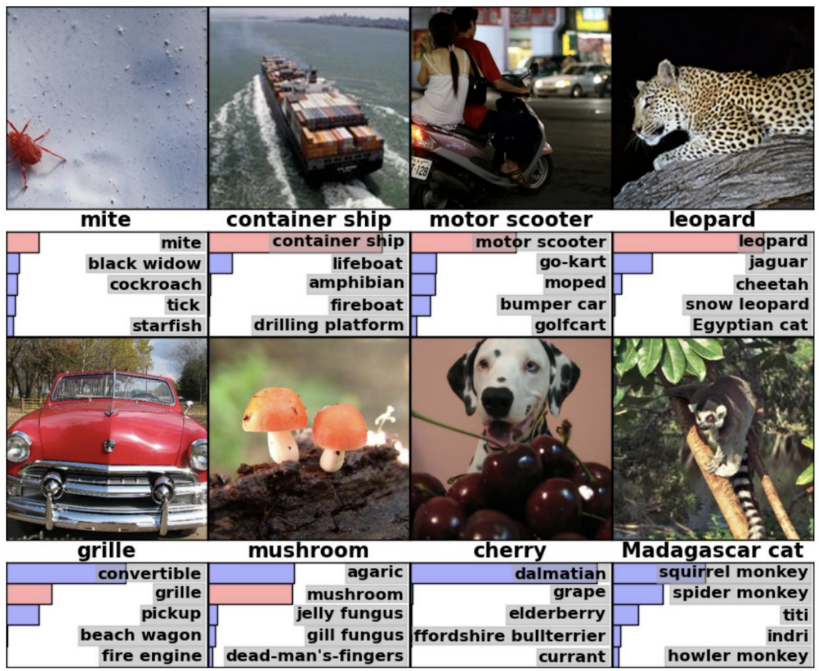
ImageNet Dataset

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *arXiv preprint arXiv:1409.0575*. [web]

Imagenet classes

ImageNet Challenge

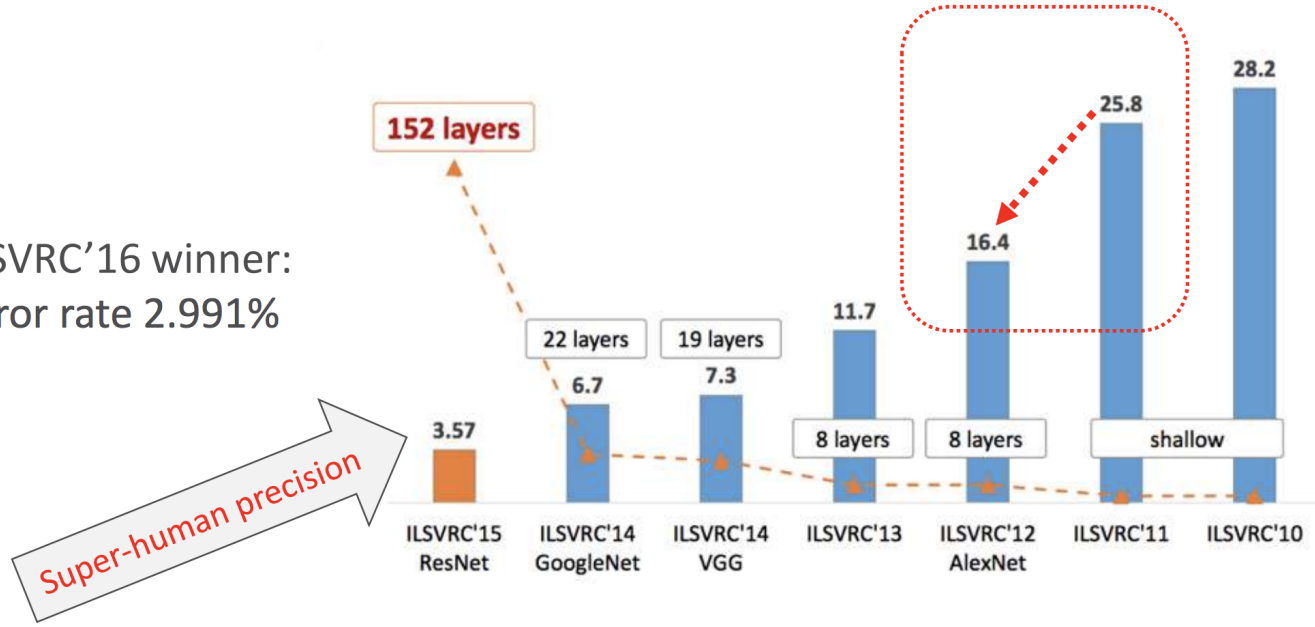
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



Revolution of Depth

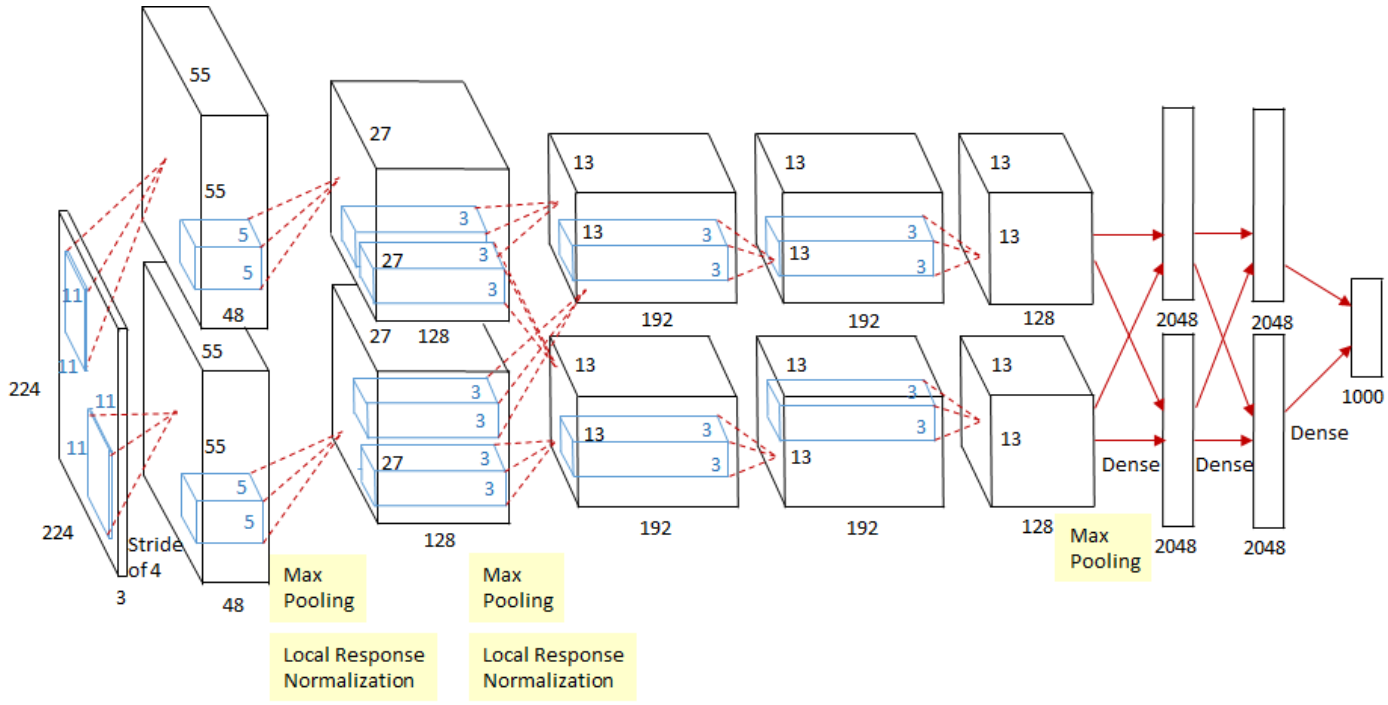
ImageNet Challenge

ILSVRC'16 winner:
Error rate 2.991%

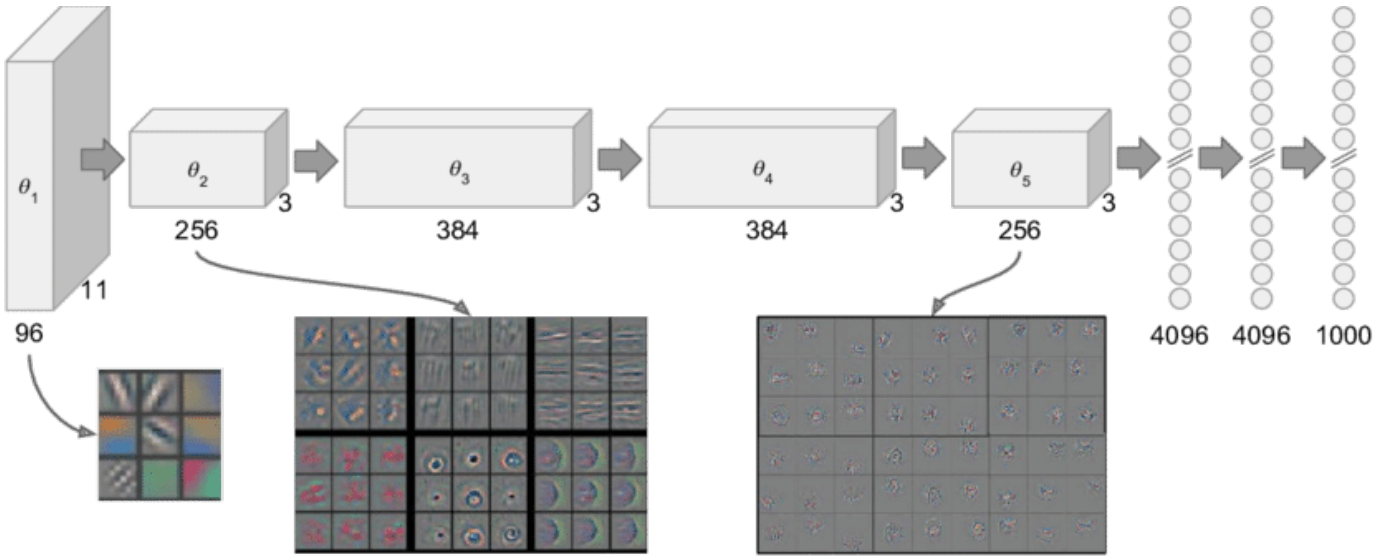


* Human-level performance: 5.1%

Alexnet


55
55
48
55
55
48
224
224
11
11
11
11
Stride
of 4
3
27
27
128
27
27
128
3
3
3
3
3
3
13
13
192
13
13
192
13
13
128
13
13
192
13
13
192
13
13
128
3
3
3
3
3
3
5
5
5
5
Max
Pooling
Local Response
Normalization
Max
Pooling
Local Response
Normalization
Max
Pooling
Dense
Dense
Dense
2048
2048
2048
2048
1000

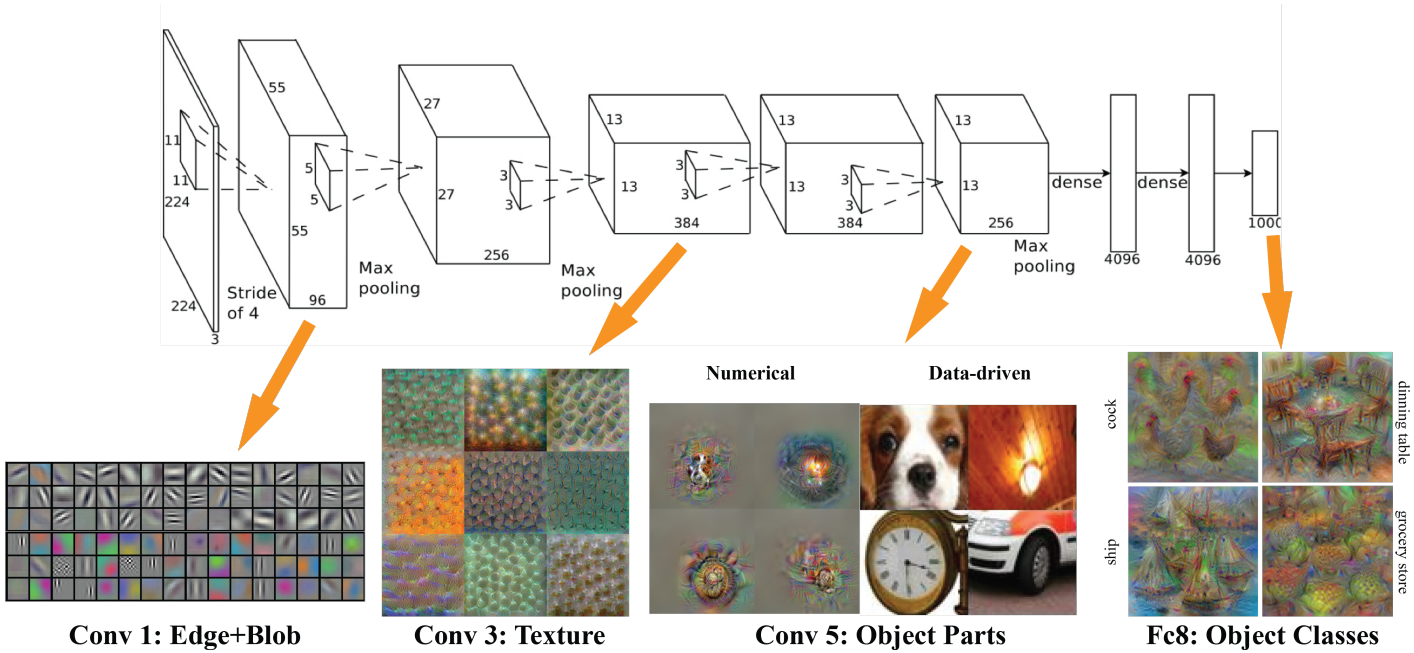
Alexnet Filter Weights



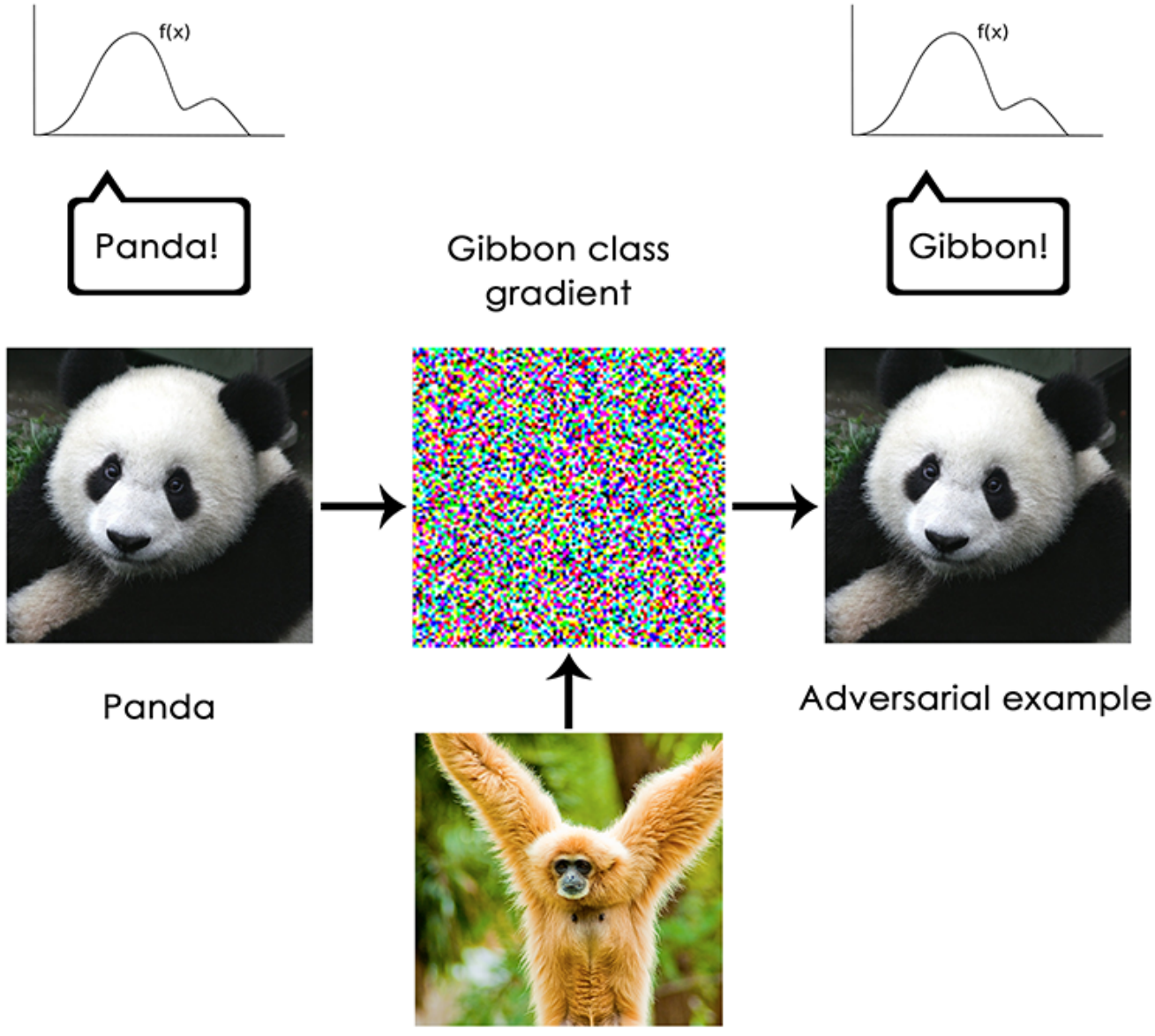
slide credit: H Scholte et al.

Which input images activate a particular neuron?

- Find an image that maximizes the activation of a single neuron (D. Wei et al.)

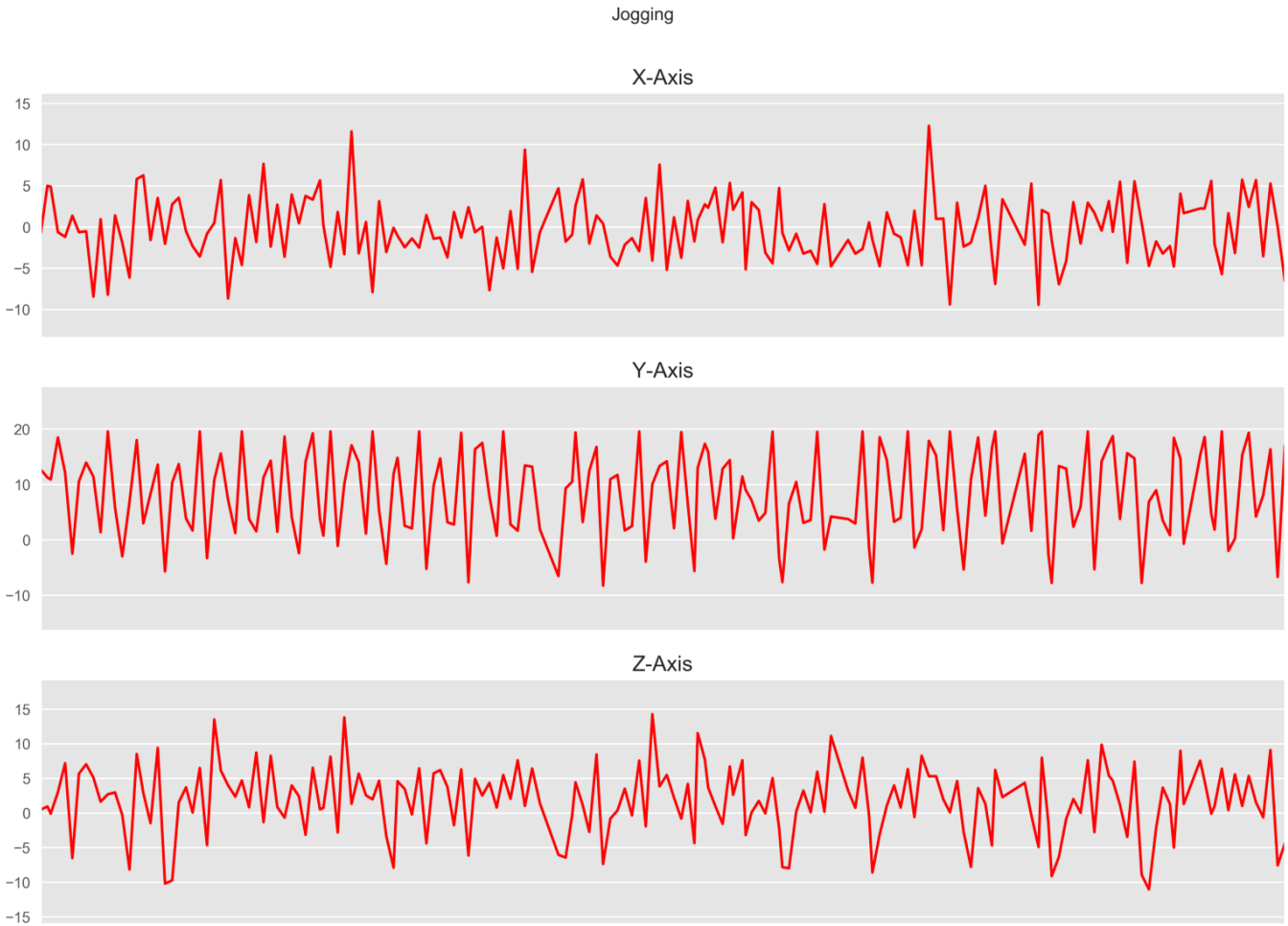


Adversarial Examples

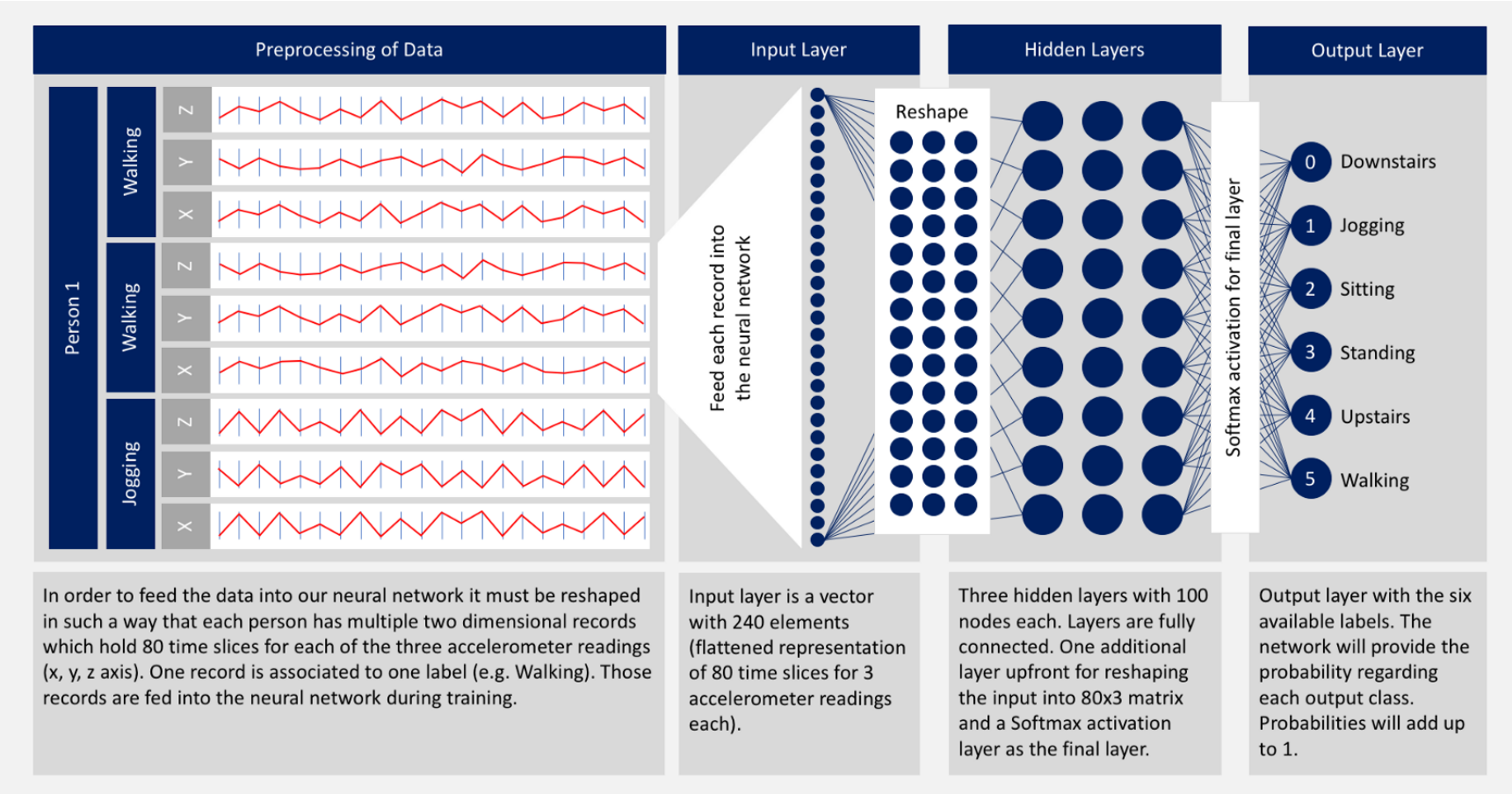


Signals and Dense Neural Nets

- Human Activity Recognition

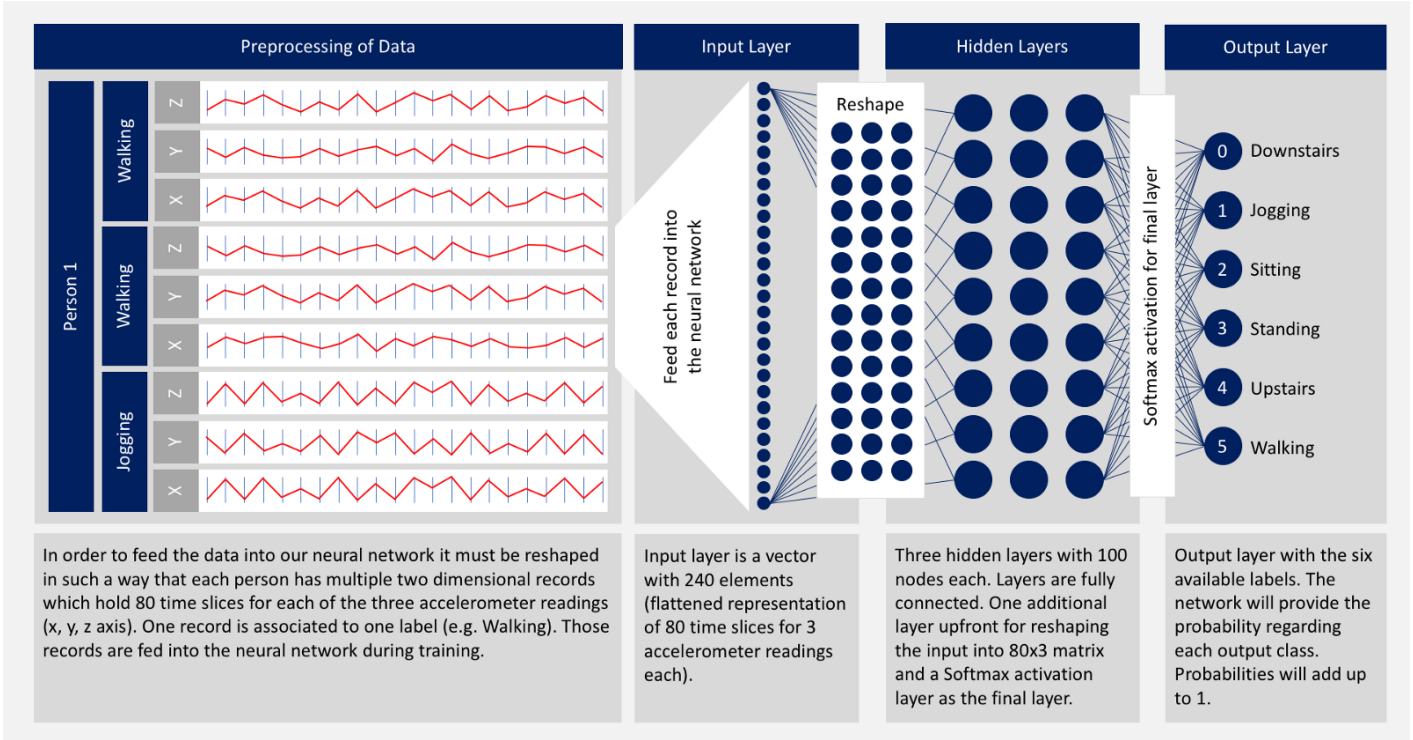


Signals and Dense Neural Nets



slide credit: N. Ackermann

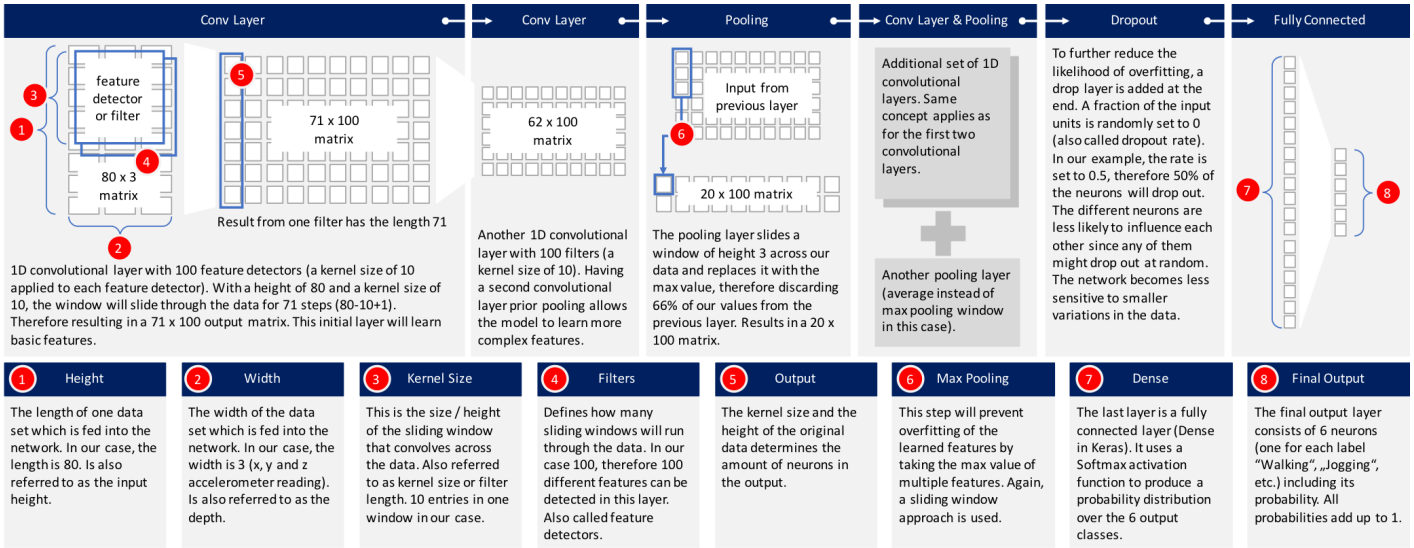
Activity Recognition Performance



- 4 dense layers, total parameters: 68,606
- Accuracy on test data: 0.76

slide credit: N. Ackermann. https://towardsdatascience.com/human-activity-recognition-har-tutorial-with-keras-and-core-ml-part-1-8c05e365dfa0

1D Convolutional Nets



- 6 layers, total parameters: 520,486

  Conv(length=10,filters=100)→Conv(10,100)→MaxPool(3)→Conv(10,160)→Conv(10,160)→GlobalAveragePooling

- Accuracy on test data: 0.92
- Dense (non-convolutional) net accuracy was 0.76 with 68,606 parameters

slide credit: N. Ackermann. https://towardsdatascience.com/human-activity-recognition-har-tutorial-with-keras-and-core-ml-part-1-8c05e365dfa0

2D Convolutional Networks for Signal Classification

- Given 1D signals $x_1[n], x_2[n]$...
- train neural networks on two dimensional representations of the signals
- examples: spectrogram, wavelet transforms...

Short-time Fourier Transform

- window signal

  e.g. $w[m] = \begin{cases} 0 & m < 0, m \geq L \\ 1 & 0 \leq m \leq L-1 \end{cases}$

- Short Time Fourier Transform (STFT)

$$X[n,k] = \sum_{m=0}^{L-1} x[n+m]w[m]e^{-j(2\pi/N)km}, \quad 0 \leq k \leq N-1 \, .$$

Short-time Fourier Transform

- window signal

  e.g. $w[m] = \begin{cases} 0 & m < 0, m \geq L \\ 1 & 0 \leq m \leq L-1 \end{cases}$

- Short Time Fourier Transform (STFT)

$$X[n,k] = \sum_{m=0}^{L-1} x[n+m]w[m]e^{-j(2\pi/N)km}, \quad 0 \leq k \leq N-1 \, .$$

- Continuous Frequency STFT

$$X[n,\lambda] = \sum_{m=0}^{L-1} x[n+m]w[m]e^{-j\lambda m},$$

Speech Commands Dataset

```
load('commandNet.mat')
```

The network is trained to recognize the following speech commands:

- "yes"
- "no"
- "up"
- "down"
- "left"
- "right"
- "on"
- "off"
- "stop"
- "go"

Load a short speech signal where a person says "stop".

```
[x,fs] = audioread('stop_command.flac');
```

Listen to the command.

```
sound(x,fs)
```

Speech Commands Dataset

Label	Count
down	1842
go	1861
left	1839
no	1853
off	1839
on	1864
right	1852
stop	1885
unknown	6483
up	1843
yes	1860

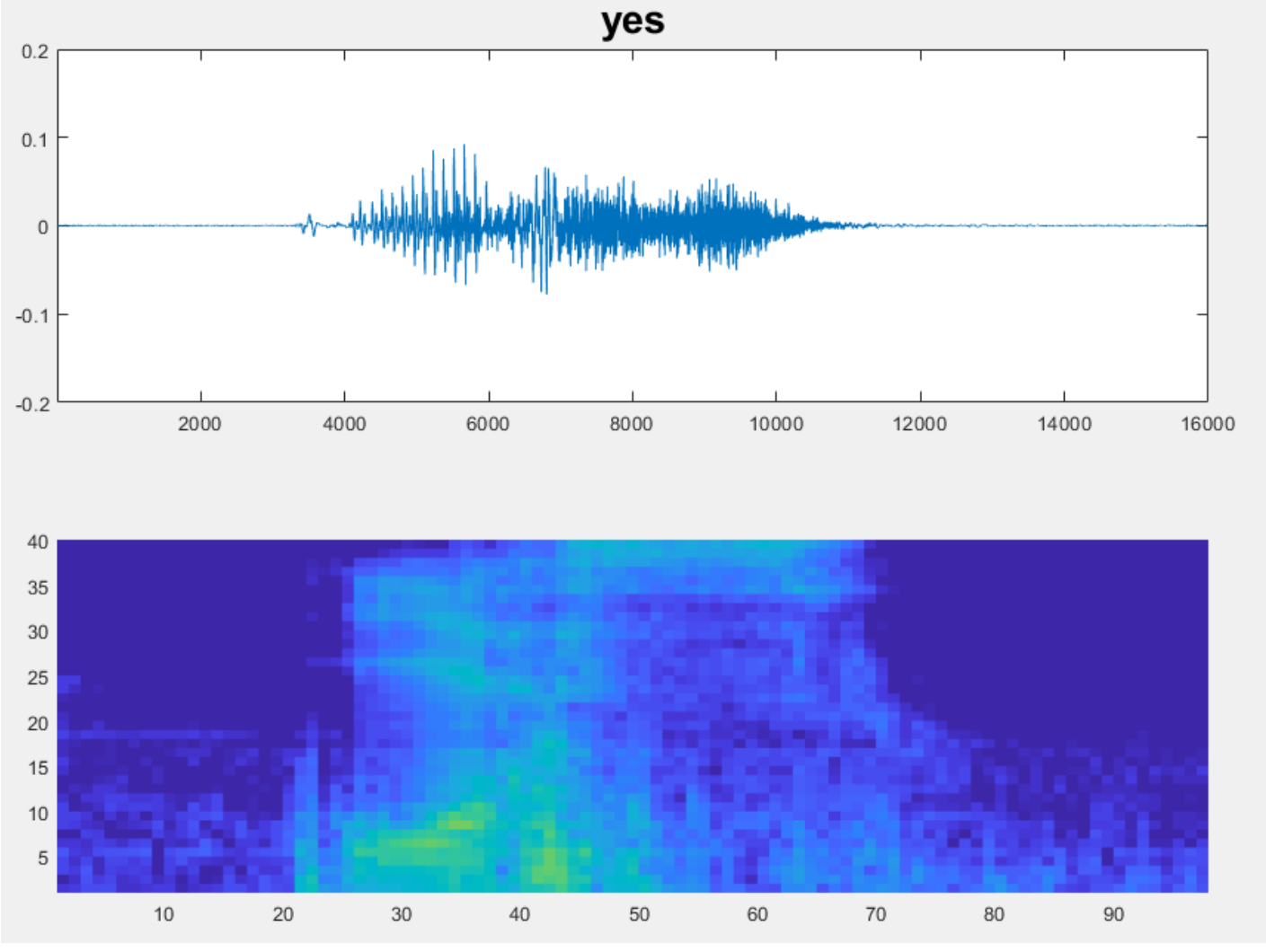
Spectrograms

```
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
numBands = 40;
```

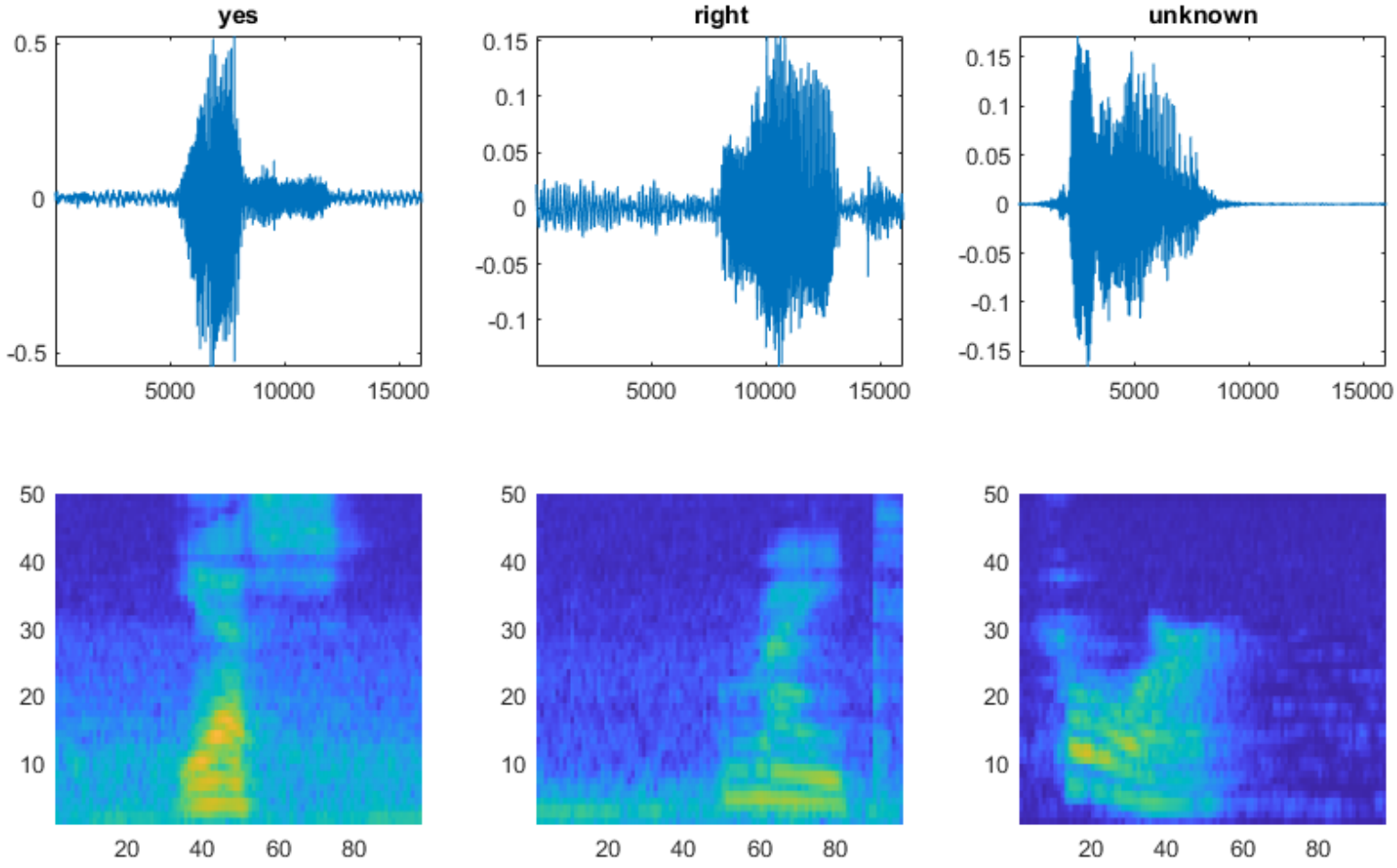
Compute the spectrograms for the training, validation, and test sets by using the supporting function `speechSpectrograms`. The `speechSpectrograms` function uses `auditorySpectrogram` for the spectrogram calculations. To obtain data with a smoother distribution, take the logarithm of the spectrograms using a small offset `epsil`.

```
epsil = 1e-6;

XTrain = speechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
XTrain = log10(XTrain + epsil);

XValidation = speechSpectrograms(adsValidation,segmentDuration,frameDuration,hopDuration,numBands);
XValidation = log10(XValidation + epsil);

XTest = speechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
XTest = log10(XTest + epsil);
```

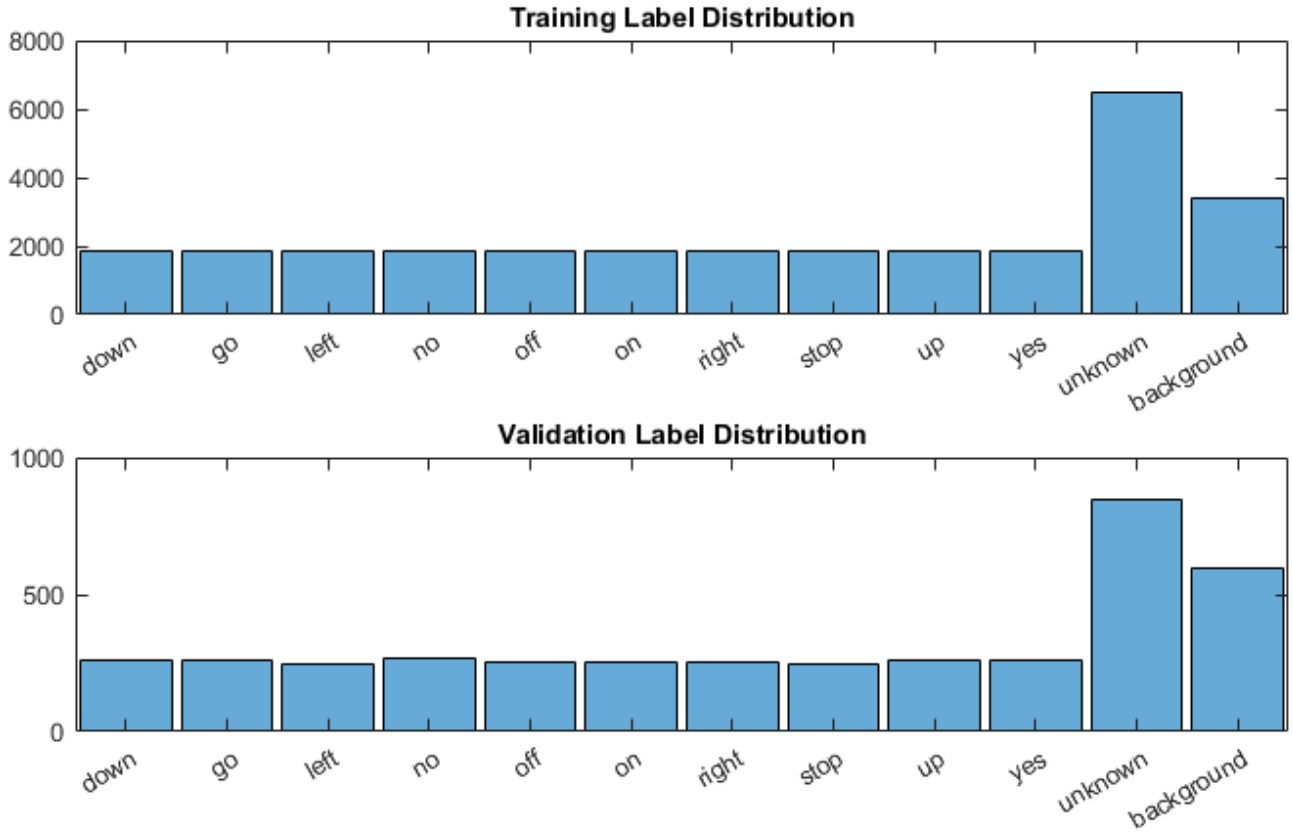
Spectrograms



Spectrograms



Training and Validation



Training and Validation

```
numF = 12;
layers = [
    imageInputLayer([numHops numBands])

    convolution2dLayer(3,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer
    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer([timePoolSize,1])

    dropoutLayer(dropoutProb)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    weightedClassificationLayer(classWeights)];
```

Training the network

Train Network

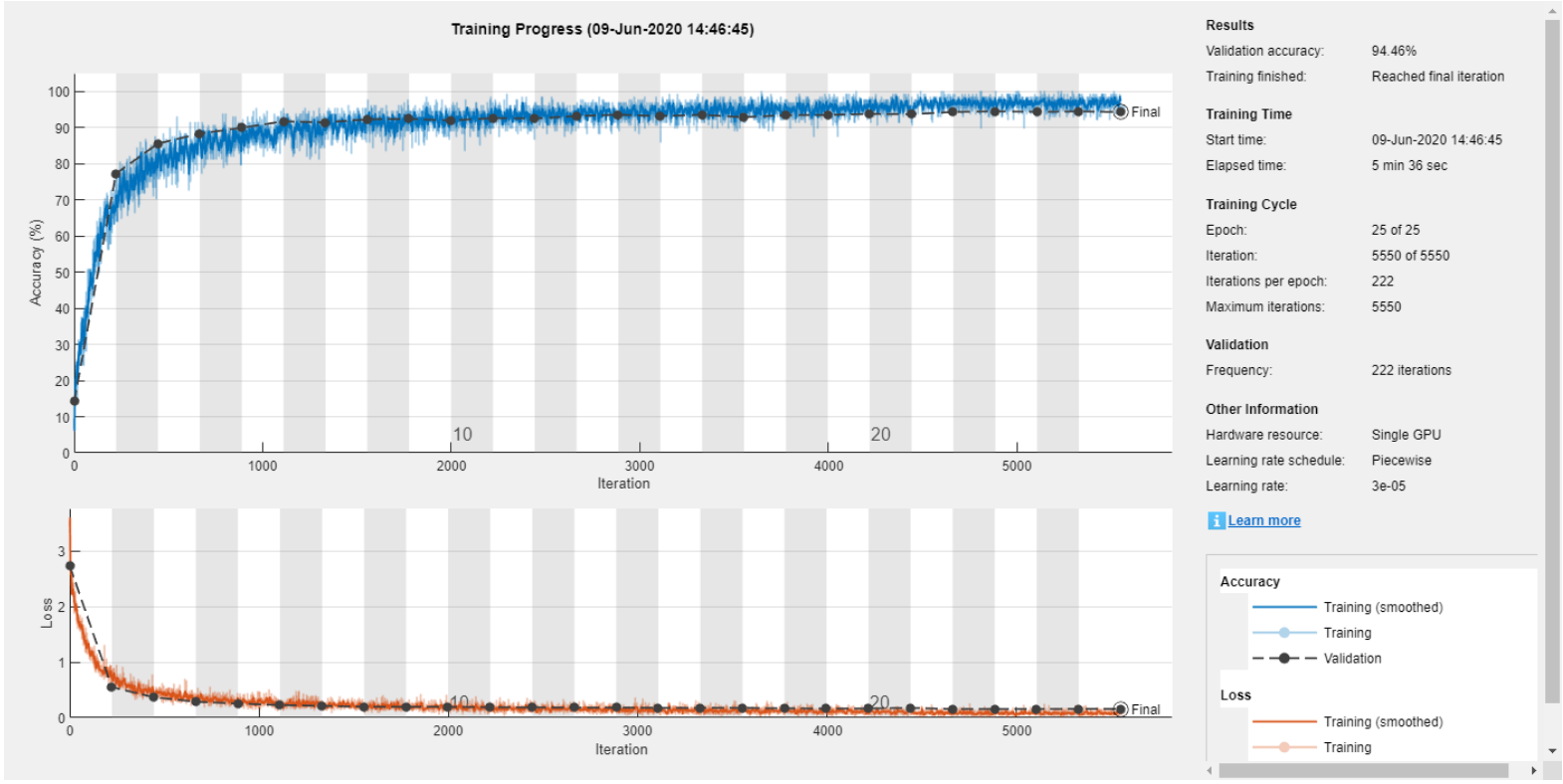
Specify the training options. Use the Adam optimizer with a mini-batch size of 128. Train for 25 epochs and reduce the learning rate by a factor of 10 after 20 epochs.

```
miniBatchSize = 128;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('adam', ...
    'InitialLearnRate',3e-4, ...
    'MaxEpochs',25, ...
    'MiniBatchSize',miniBatchSize, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose',false, ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',validationFrequency, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',20);
```

Train the network. If you do not have a GPU, then training the network can take time.

```
trainedNet = trainNetwork(augimdsTrain,layers,options);
```

Training


Training Progress (09-Jun-2020 14:46:45)
Accuracy (%)
Iteration
Final
Loss
Iteration
Final
Results
Validation accuracy: 94.46%
Training finished: Reached final iteration
Training Time
Start time: 09-Jun-2020 14:46:45
Elapsed time: 5 min 36 sec
Training Cycle
Epoch: 25 of 25
Iteration: 5550 of 5550
Iterations per epoch: 222
Maximum iterations: 5550
Validation
Frequency: 222 iterations
Other Information
Hardware resource: Single GPU
Learning rate schedule: Piecewise
Learning rate: 3e-05
Learn more
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training

Confusion Matrix