Introduction to Machine Learning
Foundations and Applications

Paul J. Atzberger
University of California Santa Barbara
Image Classification with Neural Networks

Classify Image

leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat

Neural Network (NN)
Deep neural networks are providing current state-of-the-art results.
Deep neural networks are providing current state-of-the-art results.
Deep neural networks are providing current state-of-the-art results.

ImageNet-ILSVRC challenge: 2012 ~ 16% fell to 2016 ~ 3%.
Deep neural networks are providing current state-of-the-art results.

ImageNet-ILSVRC challenge: 2012 ~ 16% fell to 2016 ~ 3%.

Popular NN architectures mimic mammalian visual cortex V1 \( \rightarrow \) Convolutional Neural Networks (CNNs).
Deep neural networks are providing current state-of-the-art results.

*ImageNet-ILSVRC challenge:* 2012 ~ 16% fell to 2016 ~ 3%.

**Popular NN architectures** mimic mammalian visual cortex V1 $\rightarrow$ Convolutional Neural Networks (CNNs).
Deep neural networks are providing current state-of-the-art results.

**ImageNet-ILSVRC challenge:** 2012 ~ 16% fell to 2016 ~ 3%.

**Popular NN architectures** mimic mammalian visual cortex V1 → Convolutional Neural Networks (CNNs).
Deep neural networks are providing current state-of-the-art results.

**ImageNet-ILSVRC challenge:** 2012 ~ 16% fell to 2016 ~ 3%.

**Popular NN architectures** mimic mammalian visual cortex V1 → Convolutional Neural Networks (CNNs).

**Local connectivity of “neuron” units** used with shared weights.
Image Classification with Neural Networks

Deep neural networks are providing current state-of-the-art results.

ImageNet-ILSVRC challenge: 2012 ~ 16% fell to 2016 ~ 3%.

Popular NN architectures mimic mammalian visual cortex V1 → Convolutional Neural Networks (CNNs).

Local connectivity of “neuron” units used with shared weights.

Deep networks → hierarchical representations “features of features.”
Convolutional Neural Networks (CNNs)
**Hubel & Wiesel 1959:** Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).
Layer Operations

**Hubel & Wiesel 1959**: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

Responses to localized regions in visual field, modeled by Gabor functions (hypothesis).

\[
s(I) = \sum_{x \in X} \sum_{y \in Y} w(x, y) I(x, y).\]

Gabor function:

\[
w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp \left( -\beta_x x^2 - \beta_y y^2 \right) \cos(f x' + \phi),
\]

\[
x' = (x - x_0) \cos(\tau) + (y - y_0) \sin(\tau),
\]

\[
y' = -(x - x_0) \sin(\tau) + (y - y_0) \cos(\tau).
\]
Layer Operations

**Hubel & Wiesel 1959**: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

Responses to localized regions in visual field, modeled by Gabor functions (hypothesis).

\[
s(I) = \sum_{x \in X} \sum_{y \in Y} w(x, y) I(x, y).
\]

Gabor function:
\[
w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp \left( -\beta_x x'^2 - \beta_y y'^2 \right) \cos(f x' + \phi),
\]
\[
x' = (x - x_0) \cos(\tau) + (y - y_0) \sin(\tau)
\]
\[
y' = -(x - x_0) \sin(\tau) + (y - y_0) \cos(\tau).
\]
Layer Operations

**Hubel & Wiesel 1959:** Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

Responses to localized regions in visual field, modeled by Gabor functions (hypothesis).

**Inspired by mammalian visual system,** engineers create NNs with local connections.

**Aim:** obtain representations by learning useful “kernels” to convolve with the input image.
Layer Operations

**Hubel & Wiesel 1959:** Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

Responses to localized regions in visual field, modeled by Gabor functions (hypothesis).

**Inspired by mammalian visual system,** engineers create NNs with local connections.

**Aim:** obtain representations by learning useful "kernels" to convolve with the input image.
Layer Operations

Hubel & Wiesel 1959: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

Responses to localized regions in visual field, modeled by Gabor functions (hypothesis).

**Inspired by mammalian visual system,** engineers create NNs with local connections.

**Aim:** obtain representations by learning useful “kernels” to convolve with the input image.

\[
s(I) = \sum_{x \in X} \sum_{y \in Y} w(x, y) I(x, y).
\]

Gabor function:
\[
w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp \left( -\beta_x x'^2 - \beta_y y'^2 \right) \cos(f x' + \phi),
\]
\[
x' = (x - x_0) \cos(\tau) + (y - y_0) \sin(\tau)
\]
\[
y' = -(x - x_0) \sin(\tau) + (y - y_0) \cos(\tau).
\]
Layer Operations

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x * w)(t) = \int x(a)w(t - a)da. \]

where \( x(t) \) is the input signal, \( w(r) \) is “kernel” weight.
**Layer Operations**

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x * w)(t) = \int x(a)w(t - a)da. \]

where \( x(t) \) is the input signal, \( w(r) \) is “kernel” weight.

**Discrete Convolution:**

\[ s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \]
Layer Operations

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x \ast w)(t) = \int x(a)w(t-a)da. \]

where \( x(t) \) is the input signal, \( w(r) \) is "kernel" weight.

**Discrete Convolution:**

\[ s(t) = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \]

Many useful operations take on this form: (Fourier transforms, filters, moving averages, estimators).
Layer Operations

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x \ast w)(t) = \int x(a)w(t - a)\,da. \]

where \( x(t) \) is the input signal, \( w(r) \) is “kernel” weight.

**Discrete Convolution:**

\[ s(t) = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \]

Many useful operations take on this form: (Fourier transforms, filters, moving averages, estimators).

**2D Convolutions: Over Images I by kernel K:**

\[ S(i, j) = (I \ast K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \] (convolution in mathematics literature)
Layer Operations

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x * w)(t) = \int x(a)w(t - a)da. \]

where \( x(t) \) is the input signal, \( w(r) \) is “kernel” weight.

**Discrete Convolution:**

\[ s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \]

Many useful operations take on this form: (Fourier transforms, filters, moving averages, estimators).

**2D Convolutions: Over Images** I by kernel \( K \):

\[ S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \quad \text{(convolution in mathematics literature)} \]

\[ S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n) \quad \text{(cross-correlation \( \rightarrow \) called a convolution in ML literature)} \]
Layer Operations

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x * w)(t) = \int x(a)w(t-a)da. \]

where \( x(t) \) is the input signal, \( w(r) \) is “kernel” weight.

**Discrete Convolution:**

\[ s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \]

Many useful operations take on this form: (Fourier transforms, filters, moving averages, estimators).

**2D Convolutions: Over Images I by kernel K:**

\[ S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \] (convolution in mathematics literature)

\[ S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n) \] (cross-correlation → called a convolution in ML literature)
Layer Operations

**Convolution** (mathematical definition for 1D):

\[ s(t) = (x \ast w)(t) = \int x(a)w(t-a)da. \]

where \( x(t) \) is the input signal, \( w(r) \) is “kernel” weight.

**Discrete Convolution:**

\[ s(t) = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \]

Many useful operations take on this form: (Fourier transforms, filters, moving averages, estimators).

**2D Convolutions: Over Images I by kernel K:**

\[ S(i, j) = (I \ast K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i-m, j-n) \]  

(\text{convolution in mathematics literature})

\[ S(i, j) = (I \ast K)(i, j) = \sum_{m} \sum_{n} I(i+m, j+n)K(m, n) \]  

(cross-correlation \( \Rightarrow \text{called a convolution in ML literature} \))

**Variant of this used in practice** for CNNs (not strictly a convolution).
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: NxM number of pixels with non-zero weights.
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: NxM number of pixels with non-zero weights.

(ii) **stride**: number of pixels to skip when computing.
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: N \times M number of pixels with non-zero weights.

(ii) **stride**: number of pixels to skip when computing.

(iii) **zero padding size**: number of zero pixels to add around the border of image.
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: N x M number of pixels with non-zero weights.

(ii) **stride**: number of pixels to skip when computing.

(iii) **zero padding size**: number of zero pixels to add around the border of image.
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: NxM number of pixels with non-zero weights.

(ii) **stride**: number of pixels to skip when computing.

(iii) **zero padding size**: number of zero pixels to add around the border of image.

[source: Theano tutorial]
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: $N \times M$ number of pixels with non-zero weights.

(ii) **stride**: number of pixels to skip when computing.

(iii) **zero padding size**: number of zero pixels to add around the border of the image.

**Example**: Convolution with 3x3 kernel, 2x2 stride, zero padding +1.
Layer Operations

Key parameters for convolution operations:

(i) **kernel size**: $N \times M$ number of pixels with non-zero weights.

(ii) **stride**: number of pixels to skip when computing.

(iii) **zero padding size**: number of zero pixels to add around the border of the image.

Example: Convolution with $3 \times 3$ kernel, $2 \times 2$ stride, zero padding $+1$.

Zero-padding important to avoid shrinkage of layers in deep networks.
Detector operation: The convolution values $S(i,j)$ are processed by a non-linearity to obtain $s(i,j) = g(S(i,j))$. Typically, ReLU + bias used $g(z) = \max(z + b, 0)$. 

ReLU + bias
Layer Operations

**Detector operation:** The convolution values $S(i,j)$ are processed by a non-linearity to obtain $s(i,j) = g(S(i,j))$. Typically, ReLU + bias is used $g(z) = \max(z + b, 0)$.

**Pooling operation:** Reduces the size of the output and helps introduce invariances like translation insensitivity.
Detector operation: The convolution values $S(i,j)$ are processed by a non-linearity to obtain $s(i, j) = g(S(i,j))$. Typically, ReLU + bias used $g(z) = \max(z + b, 0)$.

Pooling operation: Reduces size of the output and helps introduce invariances like translation in-sensitivity.

Common pooling operations:

(i) sub-sampling

(ii) averaging

(iii) max-pooling

Source: Medium
Layer Operations

Detector operation: The convolution values $S(i,j)$ are processed by a non-linearity to obtain $s(i,j) = g(S(i,j))$. Typically, ReLU + bias used $g(z) = \max(z + b, 0)$.

Pooling operation: Reduces size of the output and helps introduce invariances like translation insensitivity.

Common pooling operations:

(i) sub-sampling
(ii) averaging
(iii) max-pooling

Cross-channel pooling can be used to help learn invariances other than translation insensitivity.
**Layer Operations**

**Detector operation:** The convolution values $S(i,j)$ are processed by a non-linearity to obtain $s(i,j) = g(S(i,j))$. Typically, ReLU + bias used $g(z) = \max(z + b, 0)$.

**Pooling operation:** Reduces size of the output and helps introduce invariances like translation in-sensitivity.

**Common pooling operations:**

(i) sub-sampling

(ii) averaging

(iii) max-pooling

**Cross-channel pooling** can be used to help learn invariances other than translation in-sensitivity.
Layer Operations

**Final layer:** can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected **Feed-Forward Neural Network (FNN).**
Layer Operations

Final layer: can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected **Feed-Forward Neural Network (FNN).**

Predictions: $\hat{y}$ are typically probability of class assignments $[y]_i = y_i =$ probability image is in class $i$. 
Layer Operations

**Final layer:** can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected **Feed-Forward Neural Network (FNN).**

**Predictions:** \( \hat{y} \) are typically probability of class assignments
\( [y]_i = y_i = \) probability image is in class \( i \).
Final layer: can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected Feed-Forward Neural Network (FNN).

Predictions: $\hat{y}$ are typically probability of class assignments $[y]_i = y_i = \text{probability image is in class } i$.

The output $q \leftarrow \text{FFN}$ is turned into probability by softmax $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)} \rightarrow p(\hat{y}|x)$.

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{for } k^{th} \text{ class}$$
Layer Operations

**Final layer:** can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected **Feed-Forward Neural Network (FNN).**

**Predictions:** $\hat{y}$ are typically probability of class assignments $[y]_i = y_i =$ probability image is in class $i$.

The output $q \leftarrow \text{FFN}$ is turned into probability by softmax $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)} \rightarrow p(\hat{y}|x)$.

**Training:** back-propagation from loss $L(S)$ through the FNN + conv layers.

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{for} \quad k^{th} \text{ class}$$
Layer Operations

**Final layer:** can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected **Feed-Forward Neural Network (FNN).**

**Predictions:** \( \hat{y} \) are typically probability of class assignments \( [y]_i = y_i = \text{probability image is in class } i \).

The output \( q \leftarrow \text{FFN} \) is turned into probability by softmax \( \hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)} \rightarrow p(\hat{y}|x) \).

**Training:** back-propagation from loss \( L(S) \) through the FNN + conv layers.

**Cross-entropy loss:** \( L(S) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(\hat{y}_j^{(i)}) \), where \( y_j \) is the class 1-hot vector.

\[
1 \quad \text{hot vector} \\
0 \quad k^{\text{th}} \text{class}
\]

\[
y = \begin{bmatrix}
0 \\
1 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]
Layer Operations

**Final layer:** can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected **Feed-Forward Neural Network (FNN).**

**Predictions:** $\hat{y}$ are typically probability of class assignments $[y]_i = y_i = \text{probability image is in class } i$.

The output $\mathbf{q} \leftarrow \text{FFN}$ is turned into probability by softmax $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)} \rightarrow p(\hat{y}|x)$.

**Training:** back-propagation from loss $L(S)$ through the FNN + conv layers.

**Cross-entropy loss:** $L(S) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(\hat{y}_j^{(i)})$, where $y_j$ is the class 1-hot vector.

**Motivation:** $\tilde{L}(S) = KL(\hat{y} | y) \rightarrow KL(p(y|x) \hat{p}(x) | p(\hat{y}|x) \hat{p}(x)) = L(S) + f(Y)$, where $\hat{p}(x,y)$ is the empirical data distribution.

$1 - \text{hot vector}$

$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
Layer Operations

Final layer: can be any classifier for predictions (logistic, k-NN, SVM, NN).

Typically, fully connected Feed-Forward Neural Network (FNN).

Predictions: $\hat{y}$ are typically probability of class assignments $[y]_i = y_i = \text{probability image is in class i.}$

The output $q \leftarrow \text{FFN}$ is turned into probability by softmax $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)} \rightarrow p(\hat{y}|x)$.

Training: back-propagation from loss $L(S)$ through the FNN + conv layers.

Cross-entropy loss: $L(S) = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{10} y^{(i)}_j \log(\hat{y}^{(i)}_j)$, where $y_j$ is the class 1-hot vector.

Motivation: $\tilde{L}(S) = KL(\hat{p}(x,y) | \tilde{p}(x,\hat{y})) = KL(p(y|x)\tilde{p}(x) | p(\hat{y}|x)\tilde{p}(x)) = L(S) + f(Y)$,

$\hat{p}(x,y)$ is the empirical data distribution.

Optimization only needs $L(S)$ part, instead of full $\tilde{L}(S)$, to adjust the weights yielding $\hat{y}$.
Convolutional Neural Network: Summary

**Typical Architecture**

- **Input image** is split into “channels” -> RGB.
- **Convolution layers extract features** from the image into feature channels.
Convolutional Neural Network: Summary

**Input image** is split into “channels” -> RGB.

**Convolution layers** extract features from the image into feature channels.
Convolutional Neural Network: Summary

**Typical Architecture**

- **Input image** is split into “channels” -> RGB.
- **Convolution layers** extract features from the image into feature channels.
- **Detector operation** typically ReLU + bias.
- **Pooling** reduces entries in channel (enhances invariance to image transforms)

Source: Tutorial by Chongruo Wu
Convolutional Neural Network: Summary

**Input image** is split into “channels” -> RGB.

Con**volution layers extract features** from the image into feature channels.

**Detector operation** typically ReLU + bias.

**Pooling** reduces entries in channel (enhances invariance to image transforms)

**Deep architectures** stack to repeat.

**Fully connected layer** → predicts class.

Source: Tutorial by Chongruo Wu
CIFAR10: Image Classification
CIFAR-10: Convolutional Neural Network

Image batch
CIFAR-10: Convolutional Neural Network
CIFAR-10: Convolutional Neural Network

Image batch  Input image  Convolution layer I

3 16
CIFAR-10: Convolutional Neural Network

Image batch → Input image → Convolution layer I → ReLU + max pooling
CIFAR-10: Convolutional Neural Network

Image batch → Input image → Convolution layer I → ReLU + max pooling → Convolution layer II
CIFAR-10: Convolutional Neural Network

Image batch

Input image

Convolution layer I

ReLU + max pooling

Convolution layer II

ReLU + max pooling
CIFAR-10: Convolutional Neural Network
CIFAR-10 Dataset: consists of 60,000 images 32x32 pixels RGB with labels in 10 categories (plane, car, bird, cat, deer, dog, frog, horse, ship, truck).
CIFAR-10 Dataset: consists of 60,000 images 32x32 pixels RGB with labels in 10 categories (plane, car, bird, cat, deer, dog, frog, horse, ship, truck).

2 convolution layers + fully connected layer $\rightarrow q_i$ predicts probability of class via softmax: $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)}$. 
CIFAR-10 Dataset: consists of 60,000 images 32x32 pixels RGB with labels in 10 categories (plane, car, bird, cat, deer, dog, frog, horse, ship, truck).

2 convolution layers + fully connected layer $\Rightarrow q_i$ predicts probability of class via softmax: $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)}$.

Cross-entropy loss: $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y^{(i)}_j \log(\hat{y}^{(i)}_j), y_j$ is the class 1-hot vector.
CIFAR-10 Dataset: consists of 60,000 images 32x32 pixels RGB with labels in 10 categories (plane, car, bird, cat, deer, dog, frog, horse, ship, truck).

2 convolution layers + fully connected layer $\rightarrow q_i$ predicts probability of class via softmax: $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)}$.

Cross-entropy loss: $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} y_j^{(i)} \log(\hat{y}_j^{(i)})$, $y_j$ is the class 1-hot vector.

Optimization: Stochastic Gradient Descent with batch size 100 images.
CIFAR-10: Convolutional Neural Network

Training:

Cross-entropy loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(\hat{y}_j^{(i)}) \).

Random initial weights.

Mini-batches of size 100.

Stochastic Gradient Descent (SGD) with learning rate \( 10^{-3} \).
CIFAR-10: Convolutional Neural Network

Training:

Cross-entropy loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} y_j^{(i)} \log(\hat{y}_j^{(i)}) \).

Random initial weights.

Mini-batches of size 100.

Stochastic Gradient Descent (SGD) with learning rate \( 10^{-3} \).
CIFAR-10: Convolutional Neural Network

Training:

Cross-entropy loss: $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} y_{j}^{(i)} \log(\hat{y}_{j}^{(i)})$.

Random initial weights.

Mini-batches of size 100.

Stochastic Gradient Descent (SGD) with learning rate $10^{-3}$.

Final training loss of $7.8 \times 10^{-1}$.
CIFAR-10: Convolutional Neural Network

Hidden Layers

Diagram illustrating the flow of data through the layers of a convolutional neural network, with input and output layers specified. The diagram shows an input image batch, followed by convolutional layers with ReLU activation and max pooling, leading to a fully connected layer and finally output predictions softmax with classes such as car, bird, cat, deer, dog, frog, horse, ship, and truck.
CIFAR-10: Convolutional Neural Network

Hidden Layers
CIFAR-10: Convolutional Neural Network

Hidden Layers

Convolution Layer 1

Channel 0

Channel 1

Channel 2
CIFAR-10: Convolutional Neural Network

Hidden Layers

Convolution Layer 1

Channel 0

Channel 1

Channel 2
CIFAR-10: Convolutional Neural Network

Hidden Layers

Input image → Convolution layer 1 → ReLU + max pooling → Convolution layer 2 → ReLU + max pooling → Fully connected

Output predictions softmax:
a-plane, car, bird, cat, deer, frog, horse, ship, truck

Convolution Layer 1

Channel 0 → Channel 1 → Channel 2

Convolution Layer 2

Channel 0 → Channel 1 → Channel 2 → Channel 3

Channel 4 → Channel 5 → Channel 6 → Channel 7

Channel 8 → Channel 9 → Channel 10 → Channel 11

Channel 12 → Channel 13 → Channel 14 → Channel 15

CIFAR-10: Convolutional Neural Network

Results:

Cross-Entropy Loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} y_j^{(i)} \log(\hat{y}_j^{(i)}) \).
CIFAR-10: Convolutional Neural Network

Results:

Cross-Entropy Loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} -y_j^{(i)} \log(y_j^{(i)}) \).

Testing predictions of the neural network:
- tested on 1000 images
- tested on 2000 images
- tested on 3000 images
- tested on 4000 images
- tested on 5000 images
- tested on 6000 images
- tested on 7000 images
- tested on 8000 images
- tested on 9000 images

Tested on a total of 10000 images.
The neural network has an accuracy of 69.39%
CIFAR-10: Convolutional Neural Network

Results:

Cross-Entropy Loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} -y_j^{(i)} \log(y_j^{(i)}). \)

Predictions:

- frog: frog
- truck: truck
- deer: deer
- ship: ship
- cat: deer
- cat: dog
- car: car
- truck: ship
- deer: cat
- a-plane: a-plane
- frog: deer
- ship: ship

Testing predictions of the neural network:
- tested on 1000 images
- tested on 2000 images
- tested on 3000 images
- tested on 4000 images
- tested on 5000 images
- tested on 6000 images
- tested on 7000 images
- tested on 8000 images
- tested on 9000 images

Tested on a total of 10000 images.

The neural network has an accuracy of 69.39%
CIFAR-10: Convolutional Neural Network

Results:

Cross-Entropy Loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(y_j^{(i)}) \).

Achieves a test accuracy of 69%.

This could be improved by further adjustments of hyper-parameters, tuning of architecture, additional layers, training protocols, among other strategies.
MNIST: Digit Classification
**MNIST Dataset:** consists of 60,000 images 28x28 pixels grayscale with labels in 10 categories (digits).

2 convolution layers + fully connected layer $\rightarrow q_i$ predicts probability of class via softmax: $\hat{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)}$.

**Cross-entropy loss:** $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} -y_j^{(i)} \log(\hat{y}_j^{(i)})$, $y_j$ is the class 1-hot vector.

**Optimization:** Stochastic Gradient Descent with batch size 100 images.
**MNIST: Convolutional Neural Network**

![Diagram of a convolutional neural network with layers labeled as Image batch, Input image, Convolution layer I, ReLU + max pooling, Convolution layer II, ReLU + max pooling, Fully connected, and Output predictions softmax.]

### Training:

Cross-Entropy Loss: \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} y_j^{(i)} \log(\hat{y}_j^{(i)}) \).

Random initial weights, mini-batches of size 100.

Stochastic Gradient Descent (SGD) with learning rate \( 10^{-3} \).

**Final training loss** of \( 7.5 \times 10^{-3} \).
MNIST: Convolutional Neural Network

Hidden Layers

Convolution Layer 1

Channel 0

Convolution Layer 2

Channel 1

Channel 2

Channel 3

Channel 4

Channel 5

Channel 6

Channel 7

Channel 8

Channel 9

Channel 10

Channel 11

Channel 12

Channel 13

Channel 14

Channel 15

Image batch
Input image
Convolution layer I
ReLU + max pooling
Convolution layer II
ReLU + max pooling
Fully connected
Output predictions softmax

Paul J. Atzberger, UC Santa Barbara
Machine Learning: Foundations and Applications
http://atzberger.org/
MNIST: Convolutional Neural Network

Results:

Cross-Entropy Loss \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} y_j^{(i)} \log(\hat{y}_j^{(i)}) \)
MNIST: Convolutional Neural Network

Results:

Cross-Entropy Loss $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(\hat{y}_j^{(i)})$

Testing predictions of the neural network:
tested on 1000 images
tested on 2000 images
tested on 3000 images
tested on 4000 images
tested on 5000 images
tested on 6000 images
tested on 7000 images
tested on 8000 images
tested on 9000 images

Tested on a total of 10000 images.
The neural network has an accuracy of 99.12%
MNIST: Convolutional Neural Network

Results:

Cross-Entropy Loss \( L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(\hat{y}_j^{(i)}) \)

Testing predictions of the neural network:
- tested on 1000 images
- tested on 2000 images
- tested on 3000 images
- tested on 4000 images
- tested on 5000 images
- tested on 6000 images
- tested on 7000 images
- tested on 8000 images
- tested on 9000 images

Tested on a total of 10000 images.
The neural network has an accuracy of 99.12%
MNIST: Convolutional Neural Network

Results:

Cross-Entropy Loss $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y^{(i)}_j \log(\hat{y}^{(i)}_j)$

Achieves a test accuracy of 99.12%.

Without much tuning of the CNN, we were able to obtain very good accuracy for the MNIST dataset!
Summary
Deep neural networks are providing current state-of-the-art results.
Deep neural networks are providing current state-of-the-art results. Use of shared weights provides statistical efficiency, computationally less expensive training, and possible insights into structure in data.
Convolutional Neural Networks (CNNs): Summary

Deep neural networks are providing current state-of-the-art results.

Use of shared weights provides statistical efficiency, computationally less expensive training, and possible insights into structure in data.

Deep networks → hierarchical representations “features of features.”
Deep neural networks are providing current state-of-the-art results.

Use of shared weights provides statistical efficiency, computationally less expensive training, and possible insights into structure in data.

Deep networks $\Rightarrow$ hierarchical representations “features of features.”

Many applications of these ideas beyond image classification: language processing, reinforcement learning, data generation.