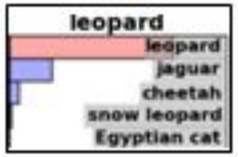
Introduction to Machine Learning

Foundations and Applications

Paul J. Atzberger
University of California Santa Barbara

Image Classification with Neural Networks

Classify Image



Neural Network (NN)

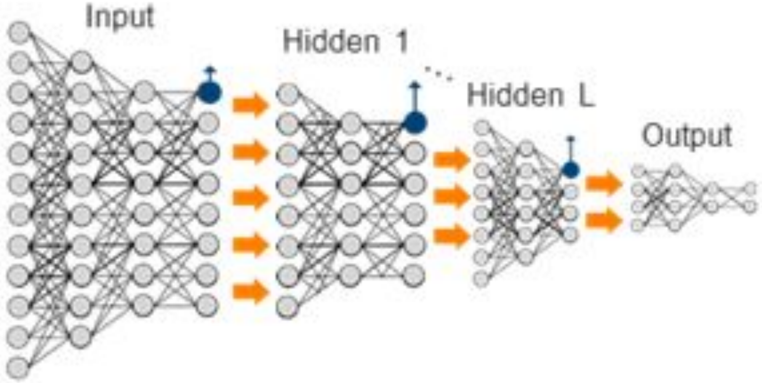
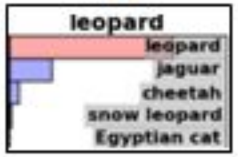
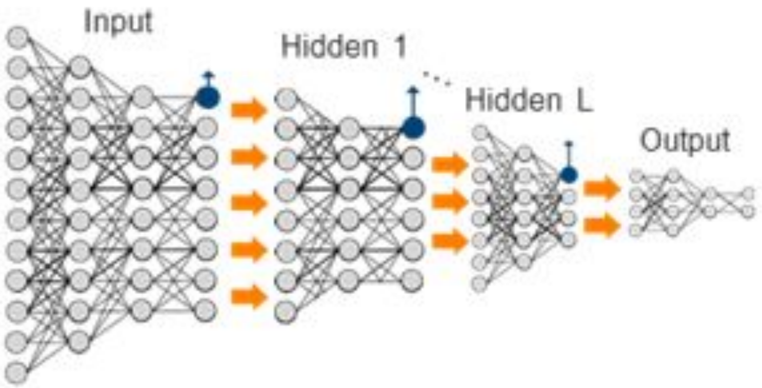


Image Classification with Neural Networks

Classify Image



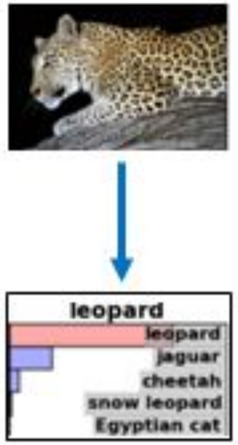
Neural Network (NN)



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

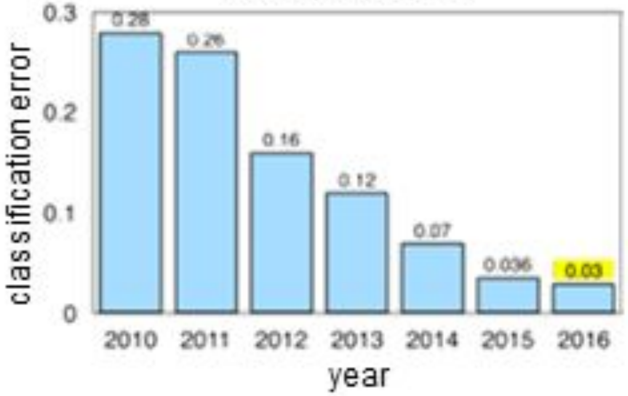
Image Classification with Neural Networks

Classify Image

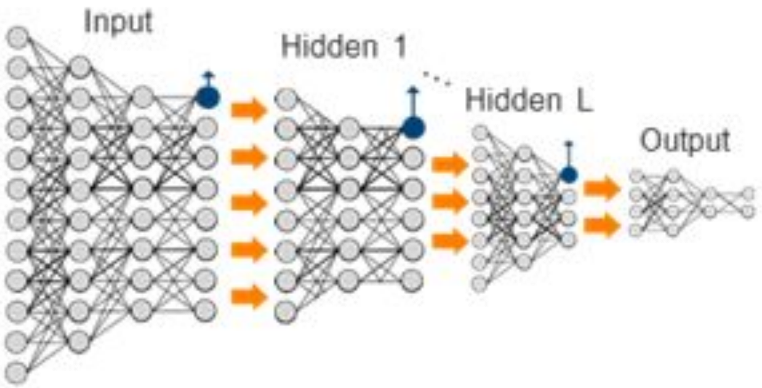


Classifier Predictions

Results: ILSVRC

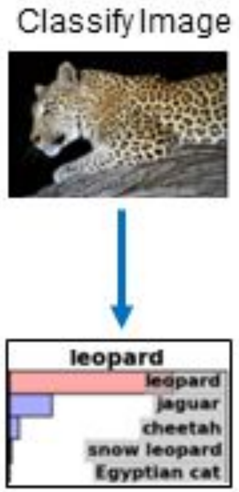


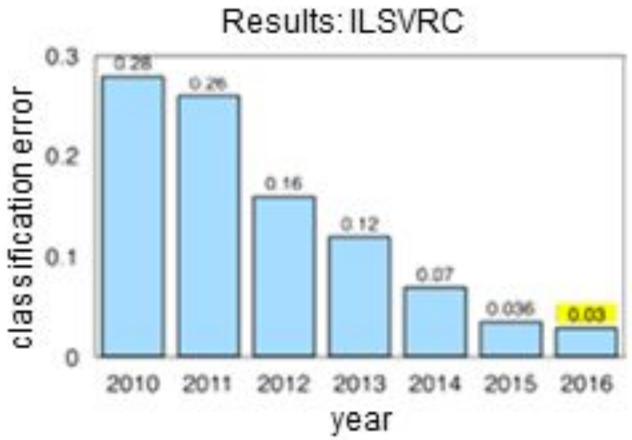
Neural Network (NN)

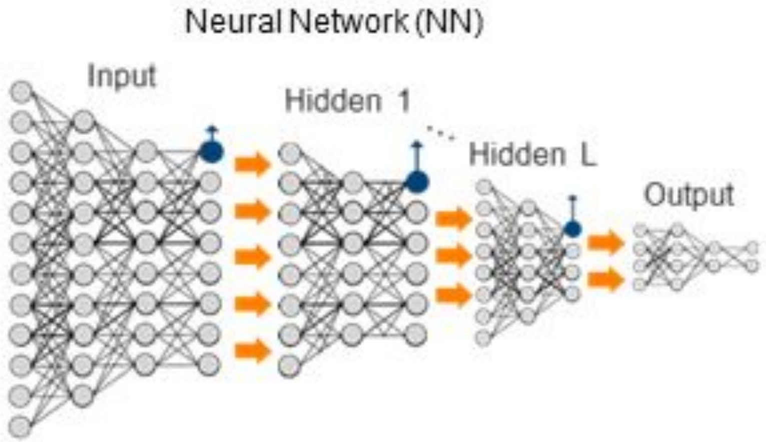


Deep neural networks are providing current state-of-the-art results.

Image Classification with Neural Networks





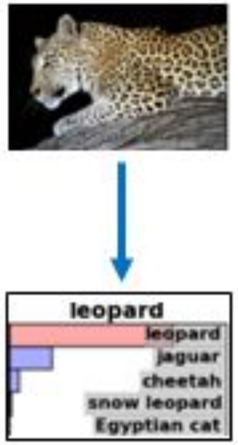


Deep neural networks are providing current state-of-the-art results.

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

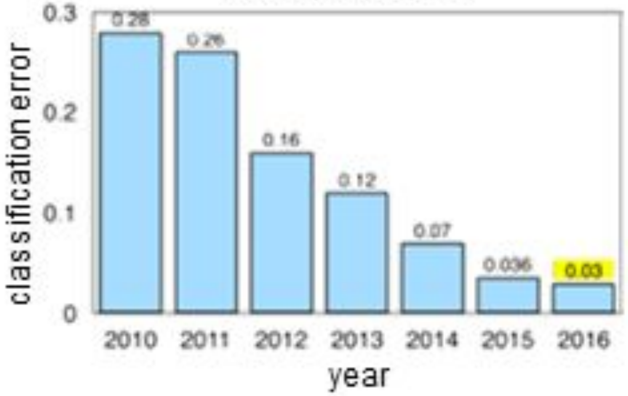
Image Classification with Neural Networks

Classify Image

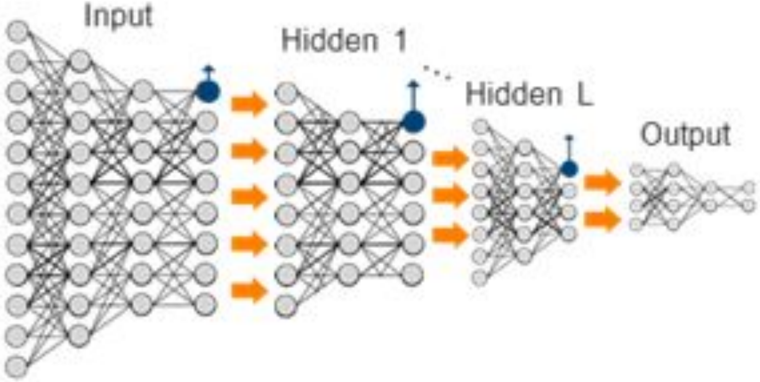


Classifier Predictions

Results: ILSVRC



Neural Network (NN)



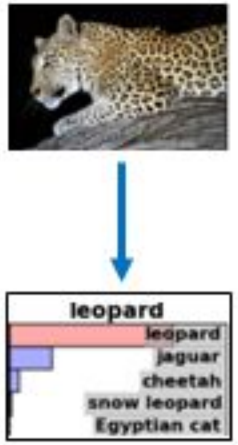
Deep neural networks are providing current state-of-the-art results.

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

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

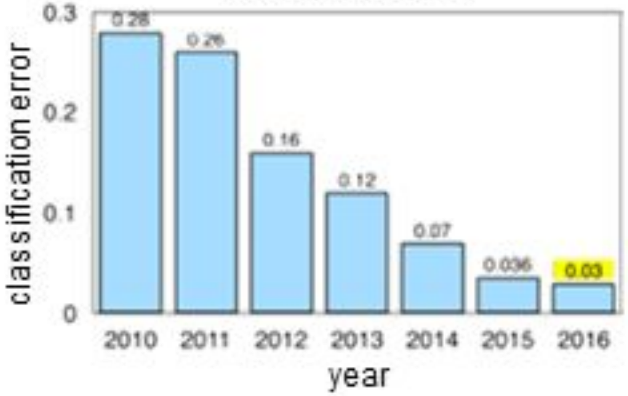
Image Classification with Neural Networks

Classify Image

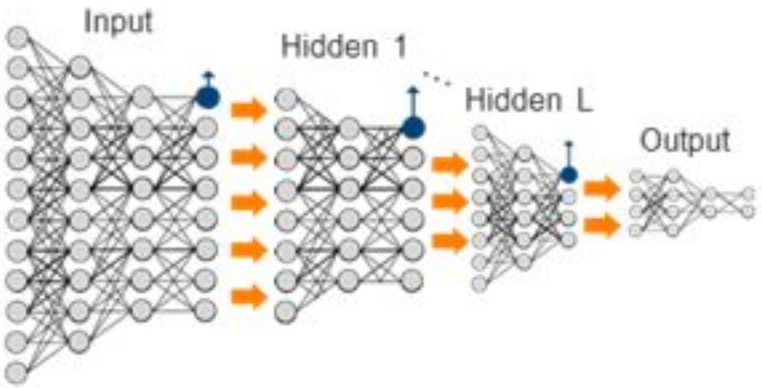


Classifier Predictions

Results: ILSVRC



Neural Network (NN)

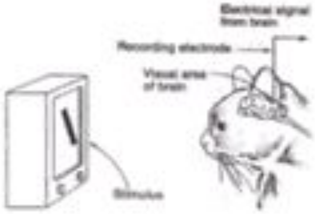


Deep neural networks are providing current state-of-the-art results.

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

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

Hubel & Wiesel 1959



Simple Cell Neuron Response

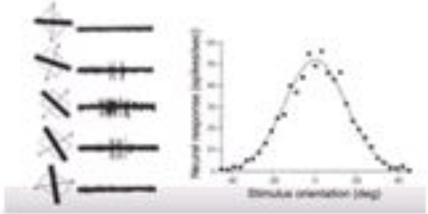
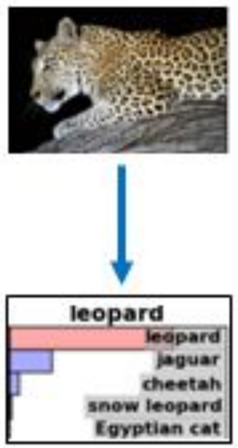


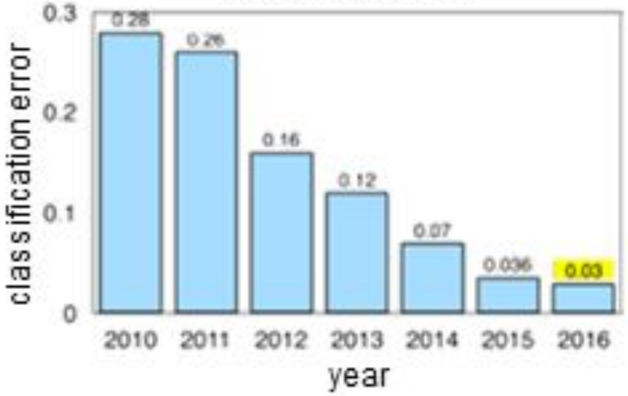
Image Classification with Neural Networks

Classify Image



Classifier Predictions

Results: ILSVRC

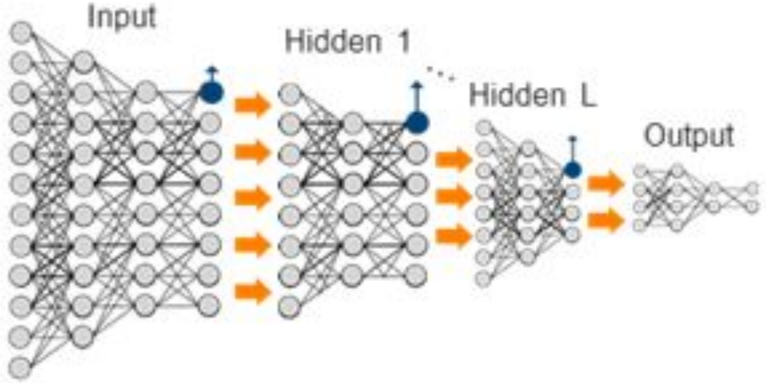


Deep neural networks are providing current state-of-the-art results.

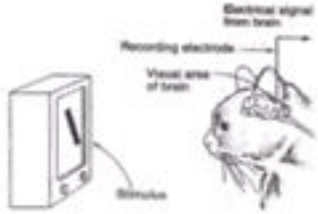
ImageNet-ILSVRC challenge: 2012 ~ 16% fell to 2016 ~ 3%.

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

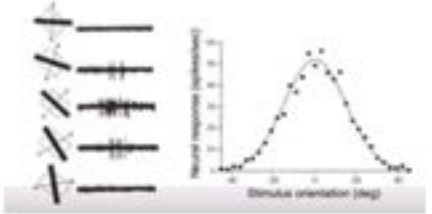
Neural Network (NN)



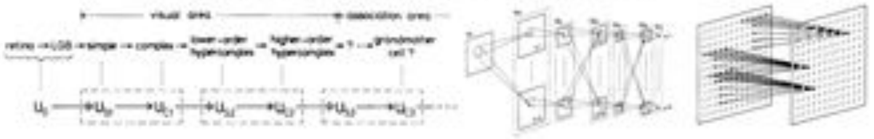
Hubel & Wiesel 1959



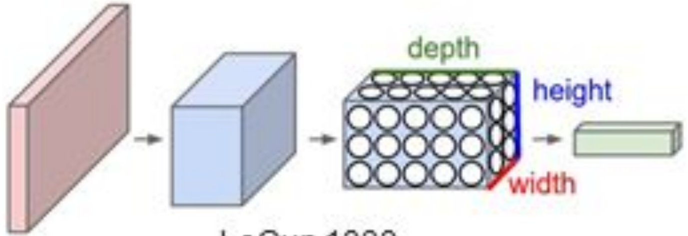
Simple Cell Neuron Response



Fukashima 1980 (Neocognitron)



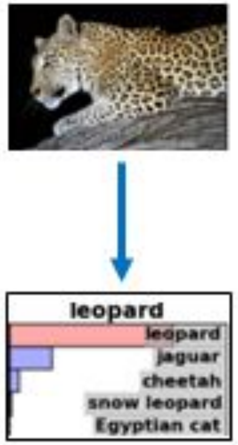
Convolutional Neural Network (CNN)



LeCun 1989

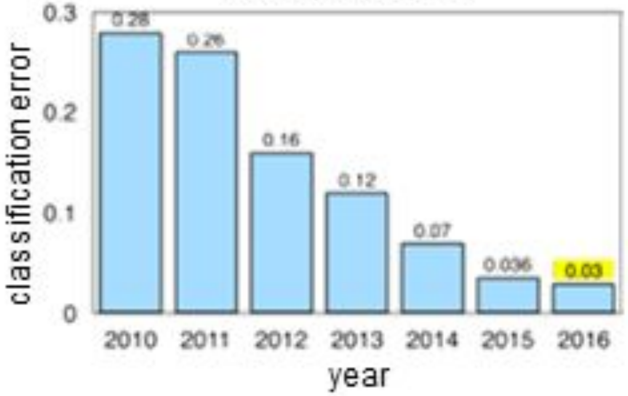
Image Classification with Neural Networks

Classify Image



Classifier Predictions

Results: ILSVRC



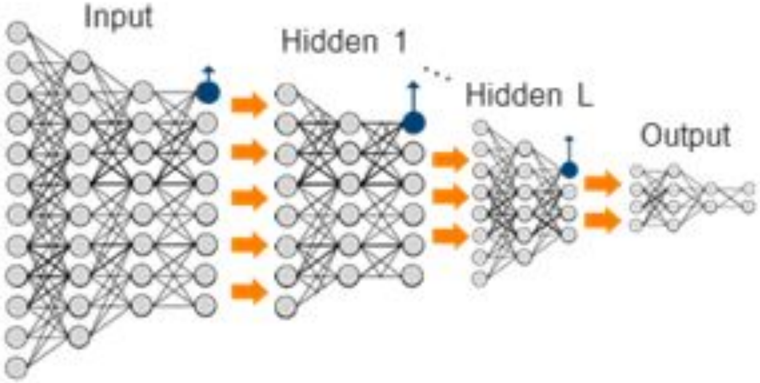
Deep neural networks are providing current state-of-the-art results.

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

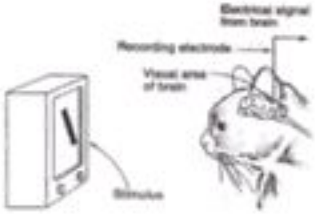
Popular NN architectures mimic mammalian visual cortex V1 → Convolutional Neural Networks (CNNs).

Local connectivity of "neuron" units used with shared weights.

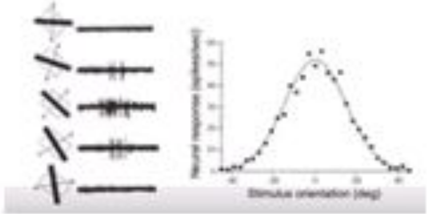
Neural Network (NN)



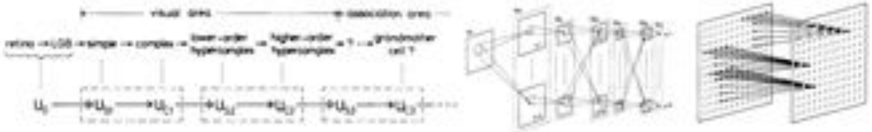
Hubel & Wiesel 1959



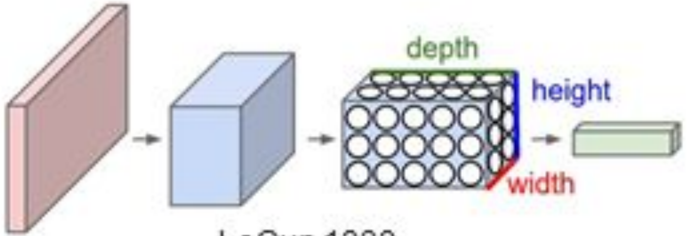
Simple Cell Neuron Response



Fukashima 1980 (Neocognitron)



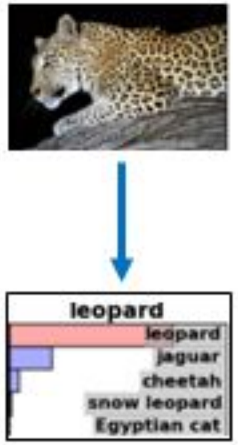
Convolutional Neural Network (CNN)



LeCun 1989

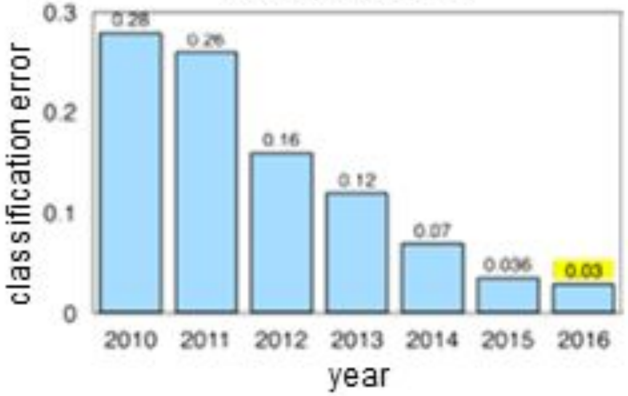
Image Classification with Neural Networks

Classify Image

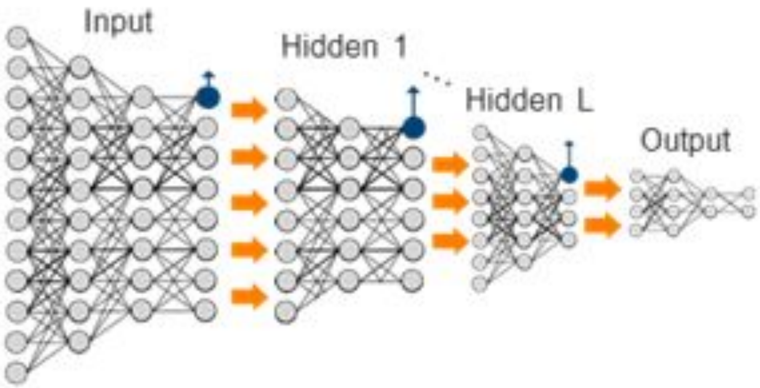


Classifier Predictions

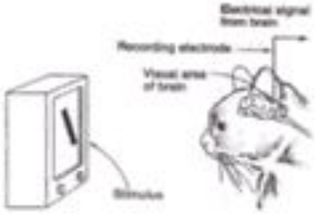
Results: ILSVRC



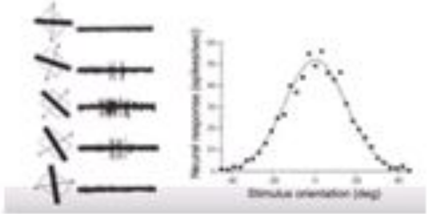
Neural Network (NN)



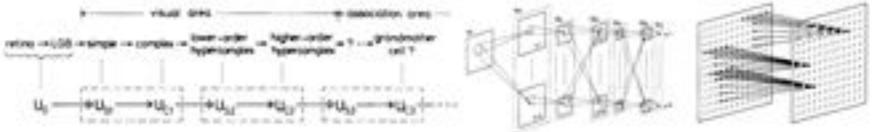
Hubel & Wiesel 1959



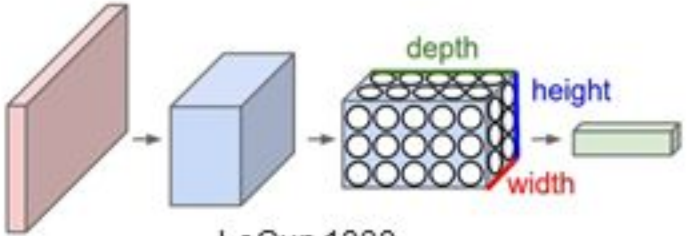
Simple Cell Neuron Response



Fukashima 1980 (Neocognitron)



Convolutional Neural Network (CNN)



LeCun 1989

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.

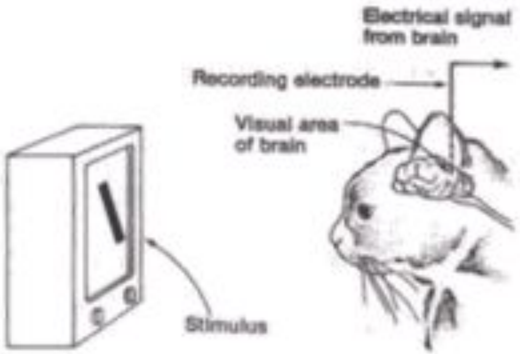
Deep networks → hierarchical representations "features of features."

Convolutional Neural Networks (CNNs)

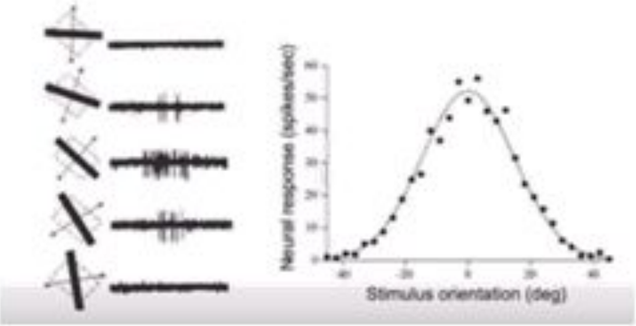
Layer Operations

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

Hubel & Wiesel 1959



Exposure to visual stimuli



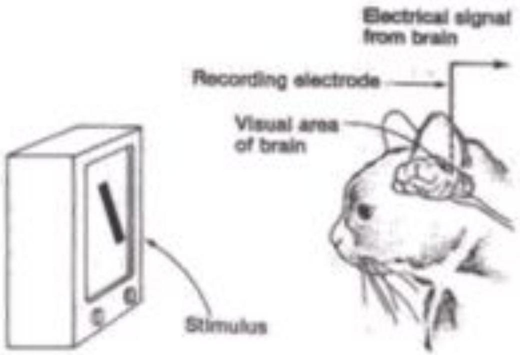
Simple Cell: Neuron Response

Layer Operations

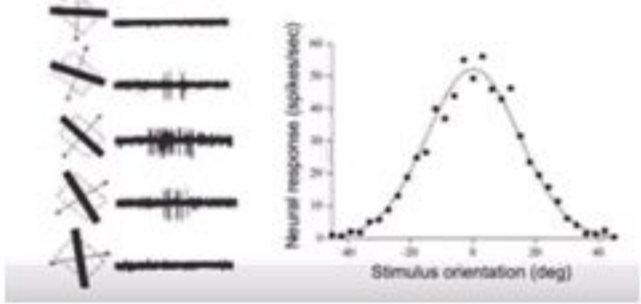
Hubel & Wiesel 1959: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

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

Hubel & Wiesel 1959



Exposure to visual stimuli



Simple Cell: Neuron Response

Models of Neuronal Responses:

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

Gabor function:

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

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

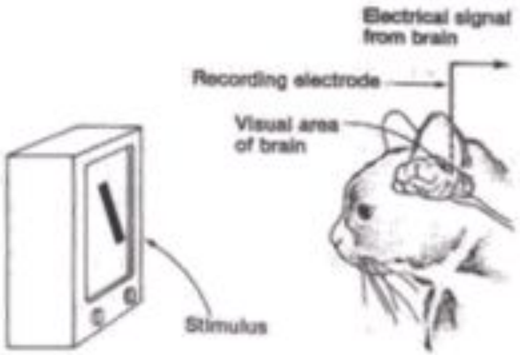
$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau).$$

Layer Operations

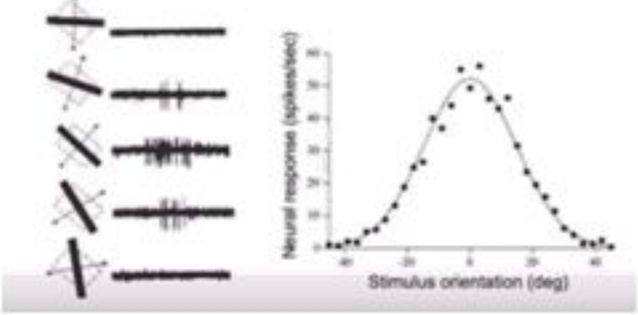
Hubel & Wiesel 1959: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

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

Hubel & Wiesel 1959



Exposure to visual stimuli



Simple Cell: Neuron Response

Models of Neuronal Responses:

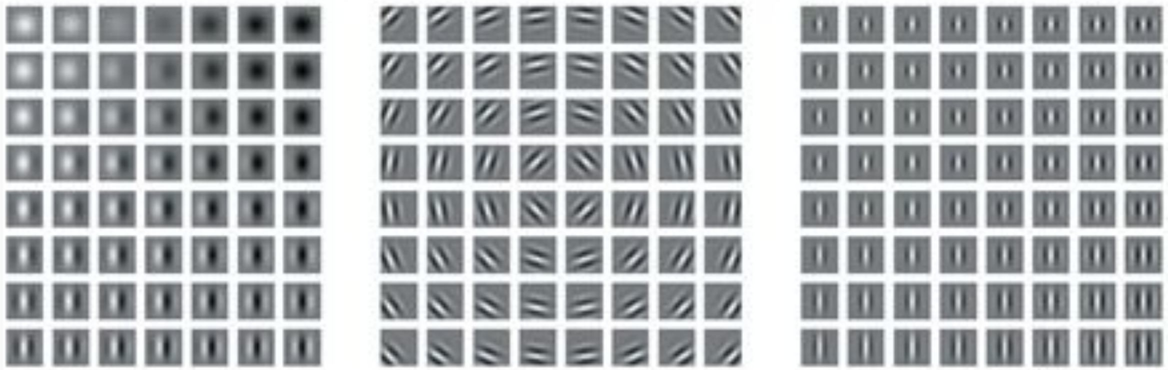
$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y).$$

Gabor function:

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

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau).$$



source: Goodfellow 2017

Layer Operations

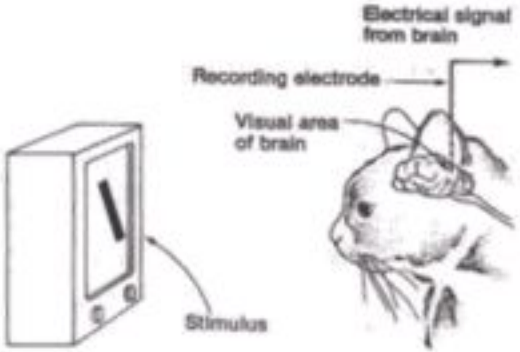
Hubel & Wiesel 1959: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

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

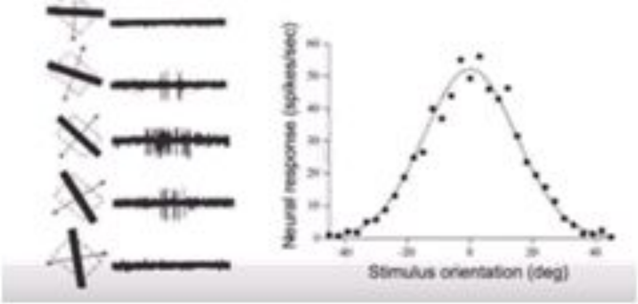
Inspired by mammalian visual system, engineers create NNs with local connections.

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

Hubel & Wiesel 1959



Exposure to visual stimuli



Simple Cell: Neuron Response

Models of Neuronal Responses:

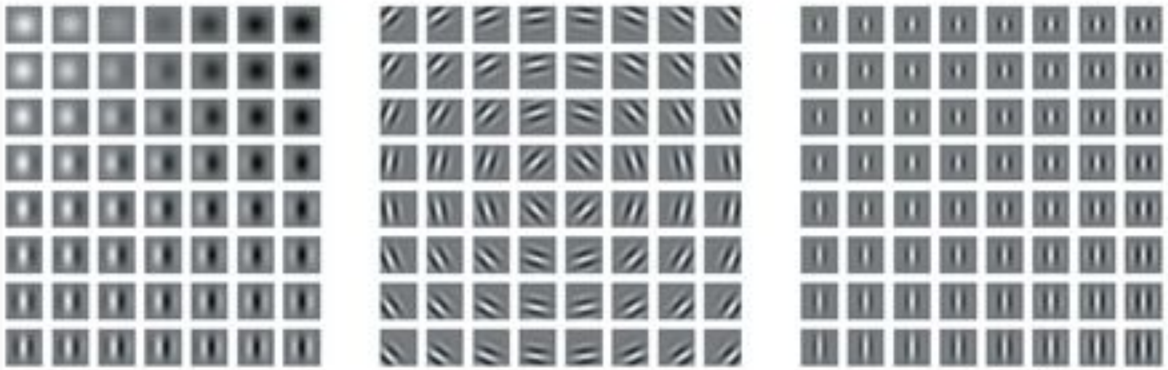
$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y).$$

Gabor function:

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

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau).$$



source: Goodfellow 2017

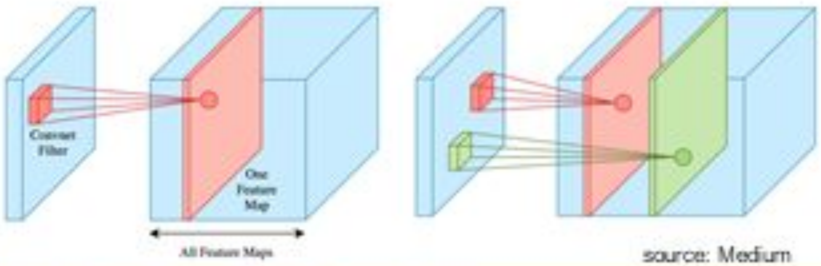
Layer Operations

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).

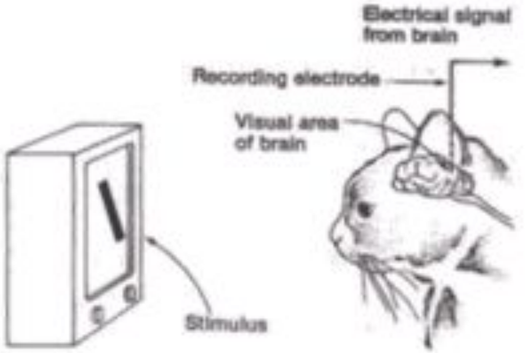
Inspired by mammalian visual system, engineers create NNs with local connections.

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

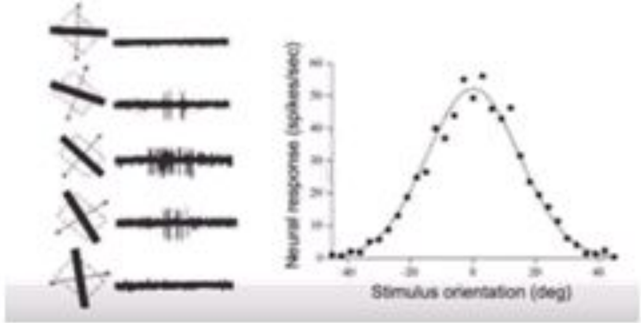


source: Medium

Hubel & Wiesel 1959



Exposure to visual stimuli



Simple Cell: Neuron Response

Models of Neuronal Responses:

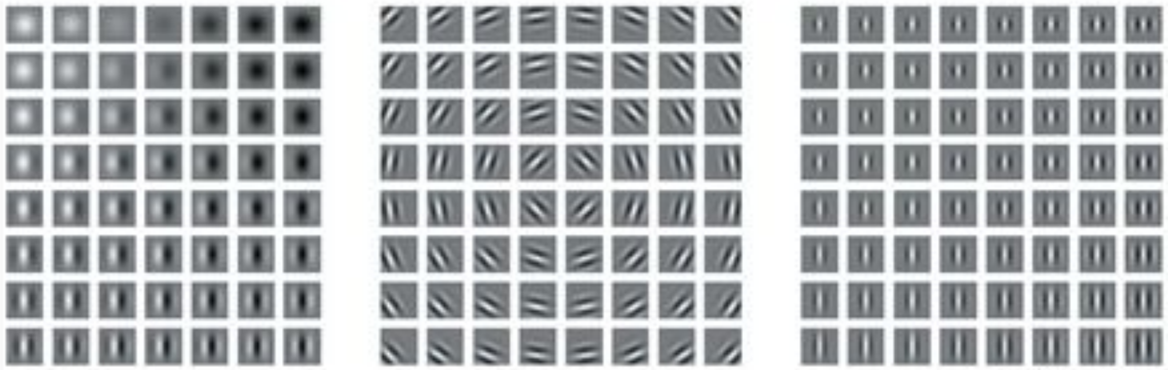
$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x,y) I(x,y).$$

Gabor function:

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

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau).$$



source: Goodfellow 2017

Layer Operations

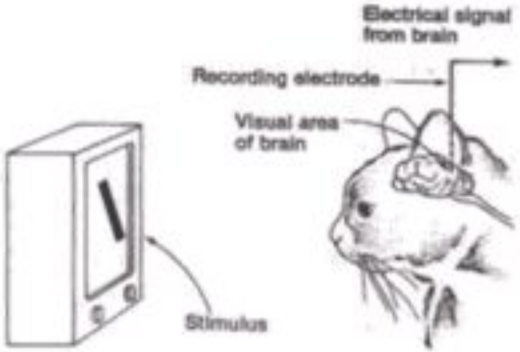
Hubel & Wiesel 1959: Experiments on cat visual cortex suggest some individual cells in V1 act as feature detectors (edges, orientation, intensity).

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

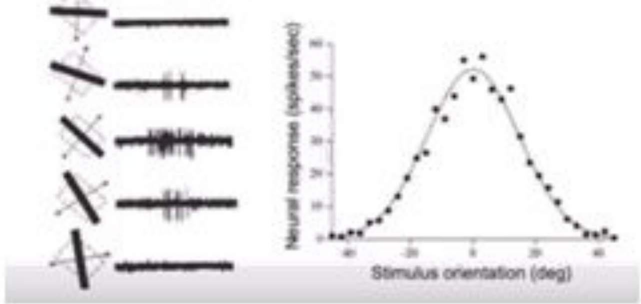
Inspired by mammalian visual system, engineers create NNs with local connections.

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

Hubel & Wiesel 1959



Exposure to visual stimuli



Simple Cell: Neuron Response

Models of Neuronal Responses:

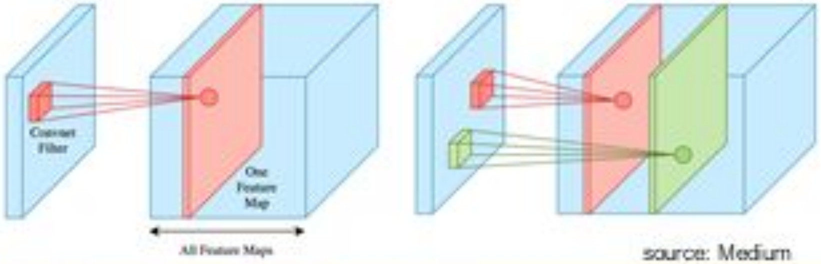
$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y).$$

Gabor function:

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

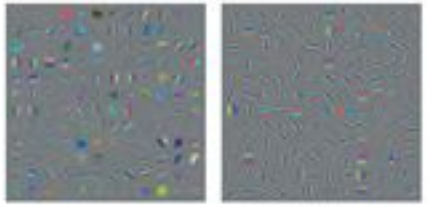
$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau).$$

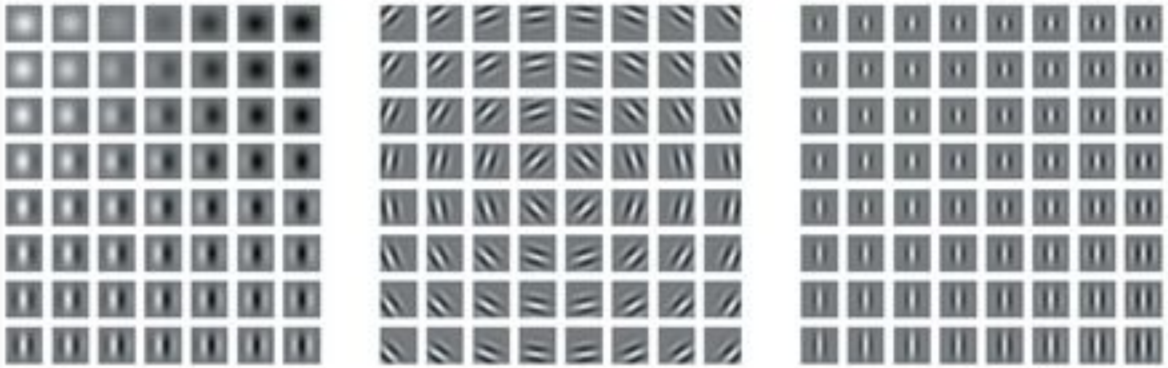


source: Medium

Kernels: trained CNN layers



source: Goodfellow 2017



source: Goodfellow 2017

Layer Operations

Convolution (mathematical definition for 1D):

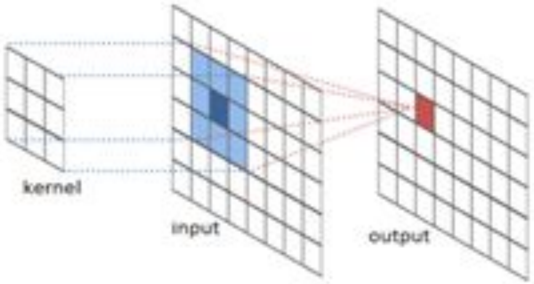
$$s(t) = (x * w)(t) = \int x(a)w(t-a)da.$$

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

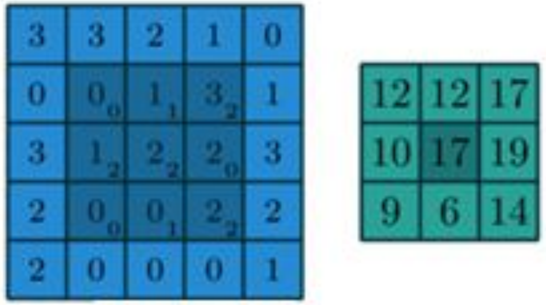
Layer Operations

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. source: http://rivertrail.github.io/RiverTrail/tutorial/



$$K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

Discrete Convolution:

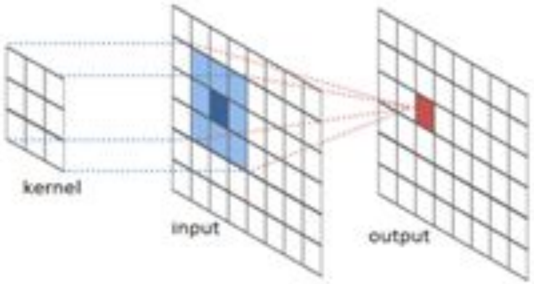
$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Layer Operations

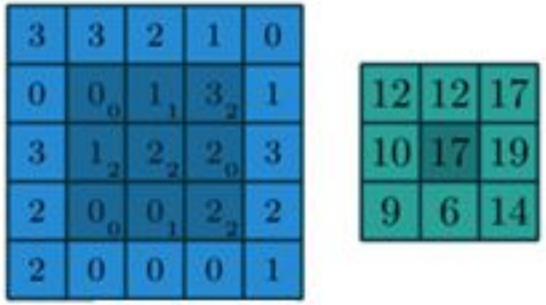
Convolution (mathematical definition for 1D):

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

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



source: http://rivertrail.github.io/RiverTrail/tutorial/



$$K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

Discrete Convolution:

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

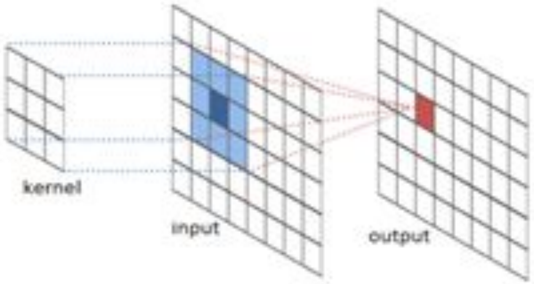
Many useful operations take on this form: (Fourier transforms, filters, moving averages, estimators).

Layer Operations

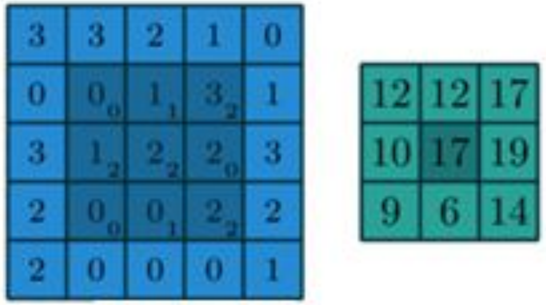
Convolution (mathematical definition for 1D):

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

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



source: http://rivertrail.github.io/RiverTrail/tutorial/



$$K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

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:

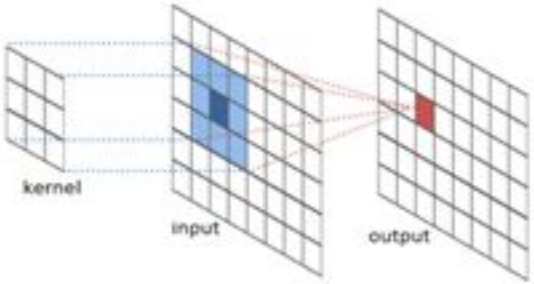
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$ (convolution in mathematics literature)

Layer Operations

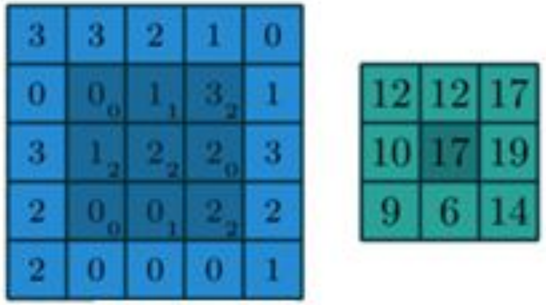
Convolution (mathematical definition for 1D):

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

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



source: http://rivertrail.github.io/RiverTrail/tutorial/



$$K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

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)

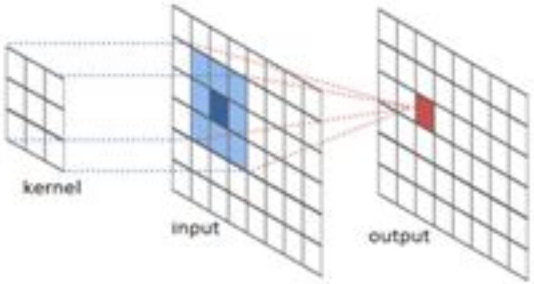
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$ (cross-correlation → called a convolution in ML literature)

Layer Operations

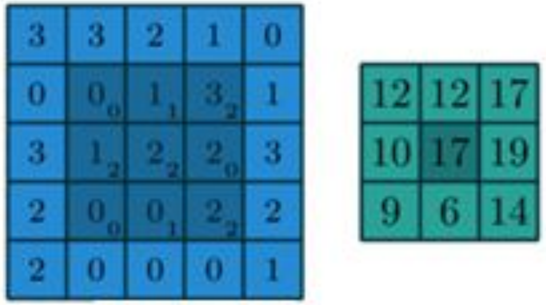
Convolution (mathematical definition for 1D):

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

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



source: http://rivertrail.github.io/RiverTrail/tutorial/



$$K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

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)

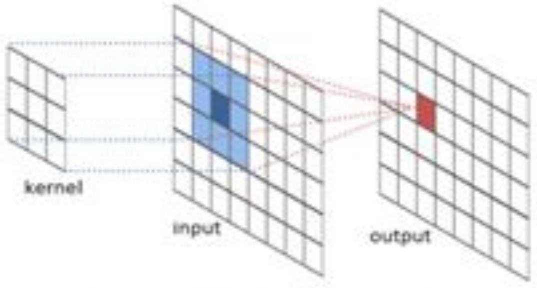
$$\boxed{S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)}$$ (cross-correlation → called a convolution in ML literature)

Layer Operations

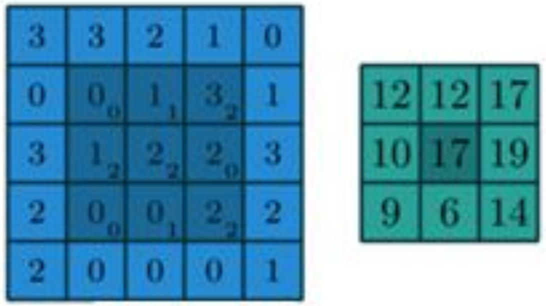
Convolution (mathematical definition for 1D):

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

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



source: http://rivertrail.github.io/RiverTrail/tutorial/



$$K = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

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)

$$\boxed{S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)}$$ (cross-correlation → called a convolution in ML literature)

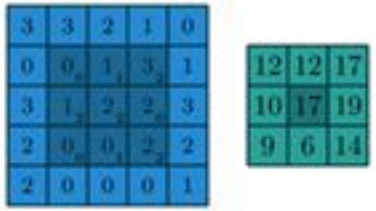
Variant of this used in practice for CNNs (not strictly a convolution).

Layer Operations

Key parameters for convolution operations:

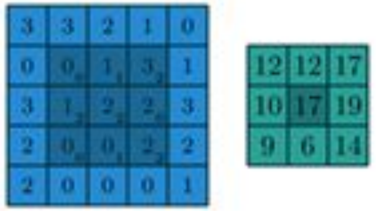
(i) **kernel size:** NxM number of pixels with non-zero weights.

Kernel Size 3x3



Layer Operations

Kernel Size 3x3



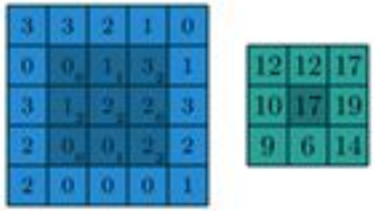
Key parameters for convolution operations:

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

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

Layer Operations

Kernel Size 3x3



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.

Layer Operations

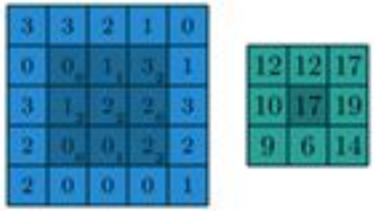
Key parameters for convolution operations:

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

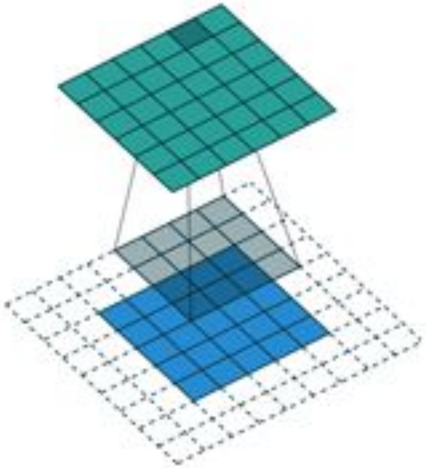
(ii) **stride:** number of pixels to skip when computing.

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

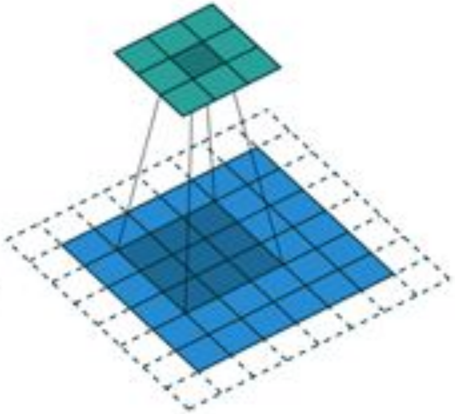
Kernel Size 3x3



Stride 1x1



Stride 2x2



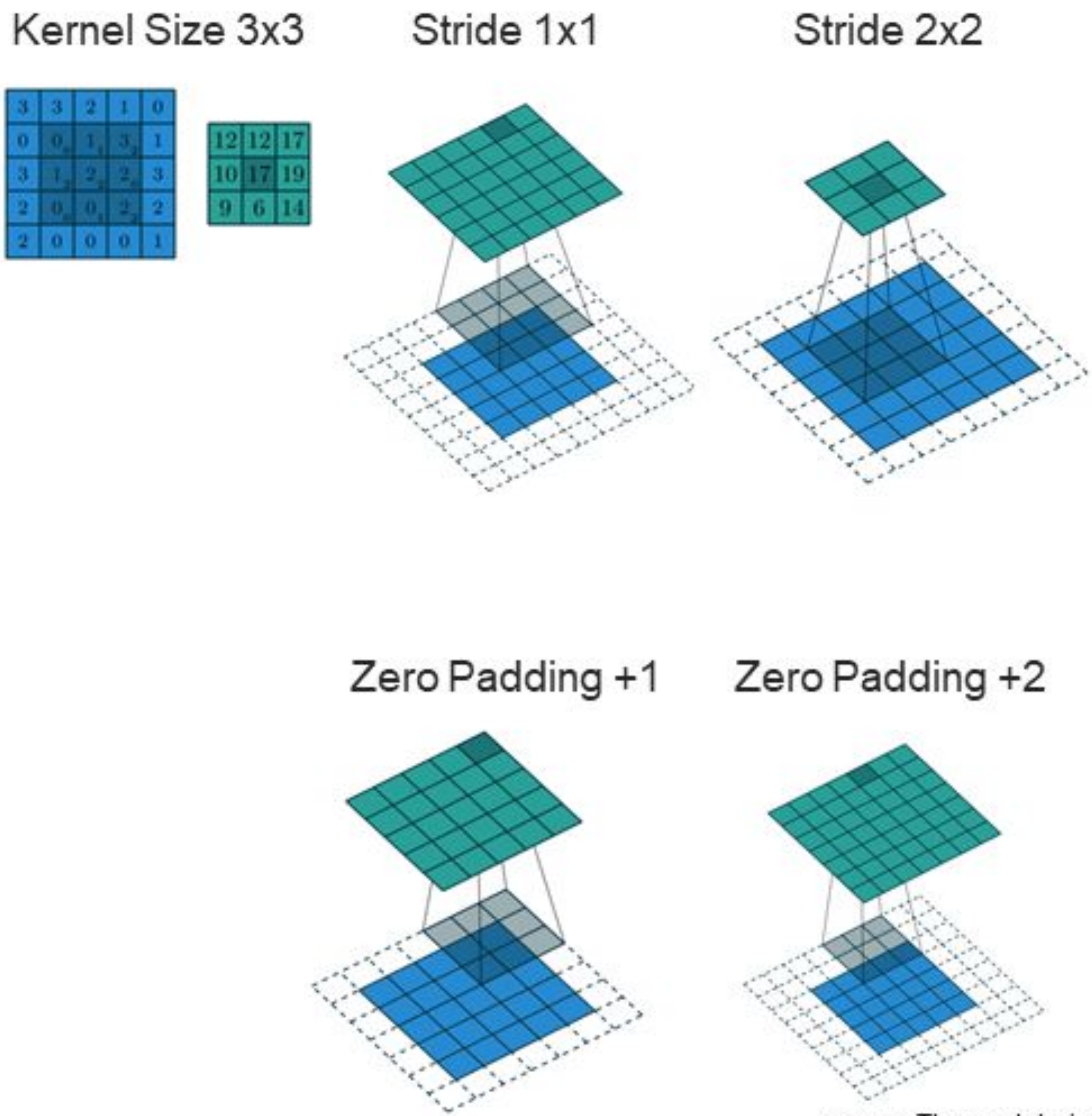
Layer Operations

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

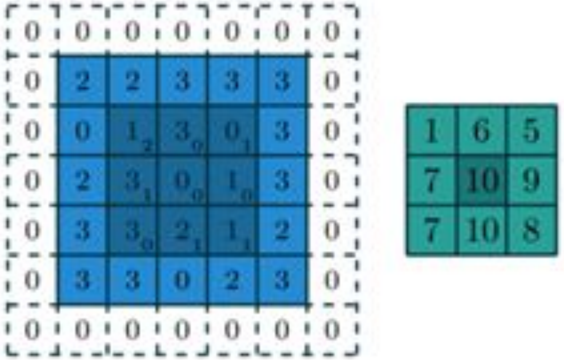
Key parameters for convolution operations:

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

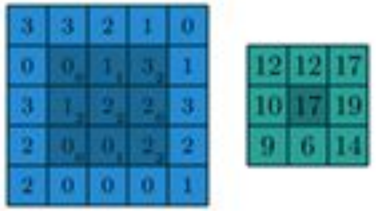
(ii) **stride:** number of pixels to skip when computing.

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

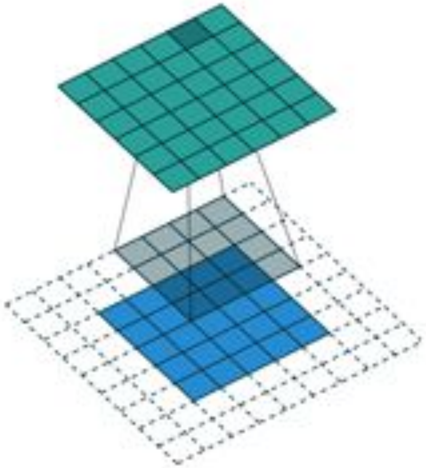
Example: Convolution with 3x3 kernel, 2x2 stride, zero padding +1.



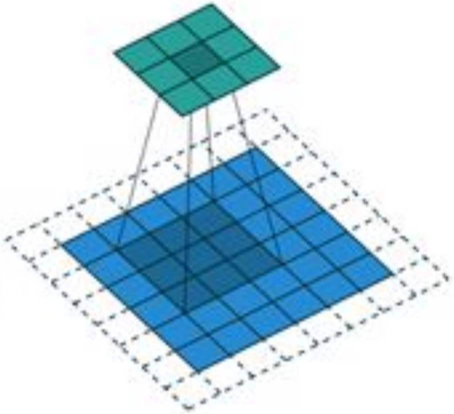
Kernel Size 3x3



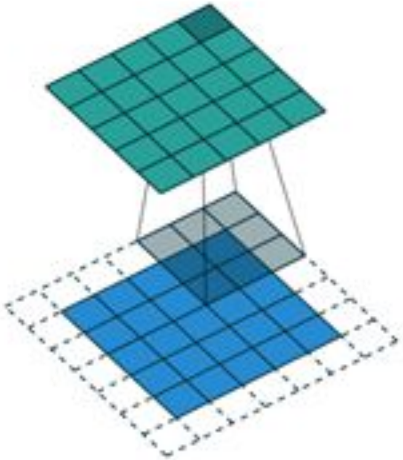
Stride 1x1



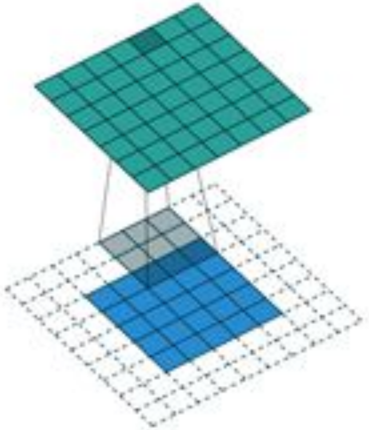
Stride 2x2



Zero Padding +1



Zero Padding +2



source: Theano tutorial.

Layer Operations

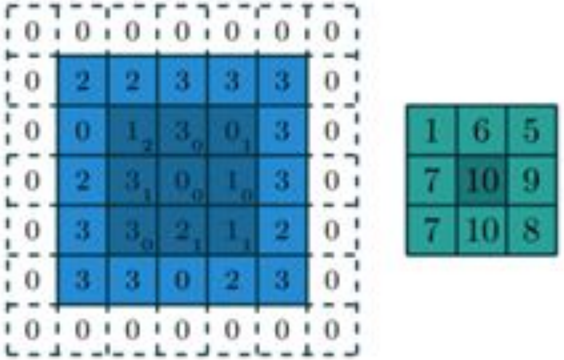
Key parameters for convolution operations:

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

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

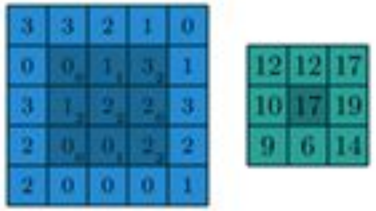
(iii) **zero padding size:** number of zero pixels to add around the border of image.

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

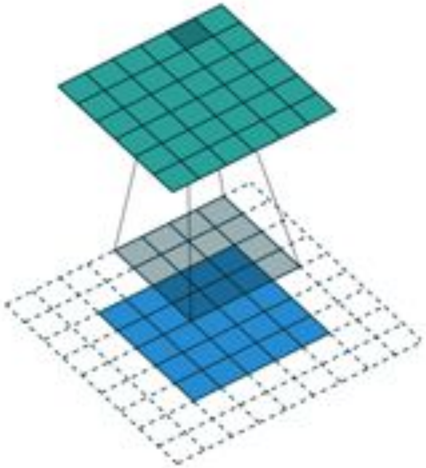


Zero-padding important to avoid shrinkage of layers in deep networks.

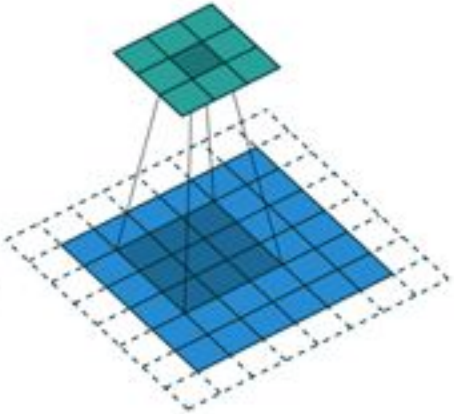
Kernel Size 3x3



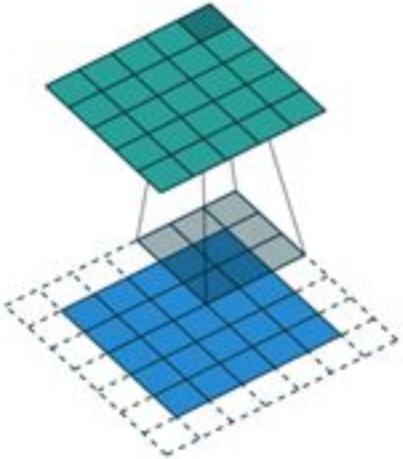
Stride 1x1



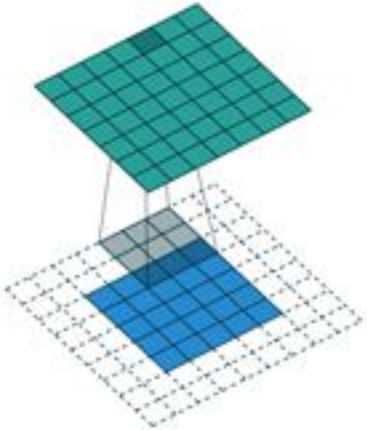
Stride 2x2



Zero Padding +1



Zero Padding +2



source: Theano tutorial.

Layer Operations

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

ReLU + bias

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

Layer Operations

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

ReLU + bias

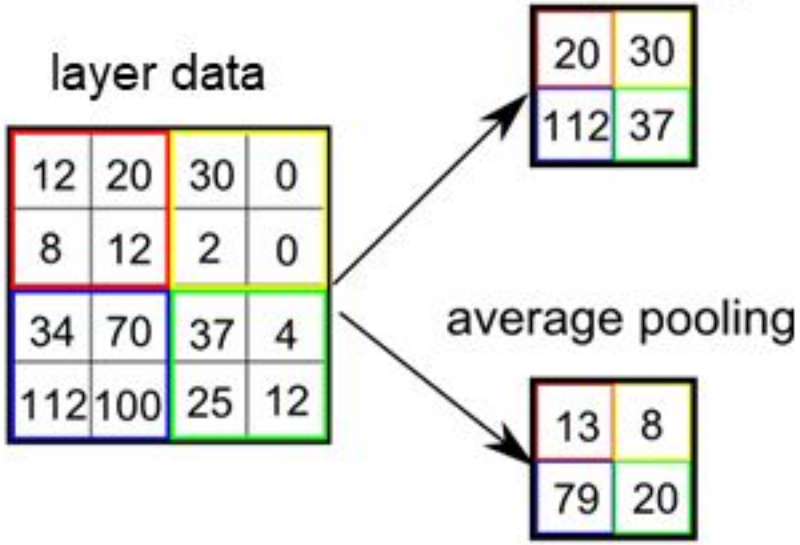
Pooling operation: Reduces size of the output and helps introduce invariances like translation in-sensitivity.

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 $s(i,j) = g\big(S(i,j)\big)$. Typically, ReLU + bias used g(z) = max(z + b,0).

ReLU + bias

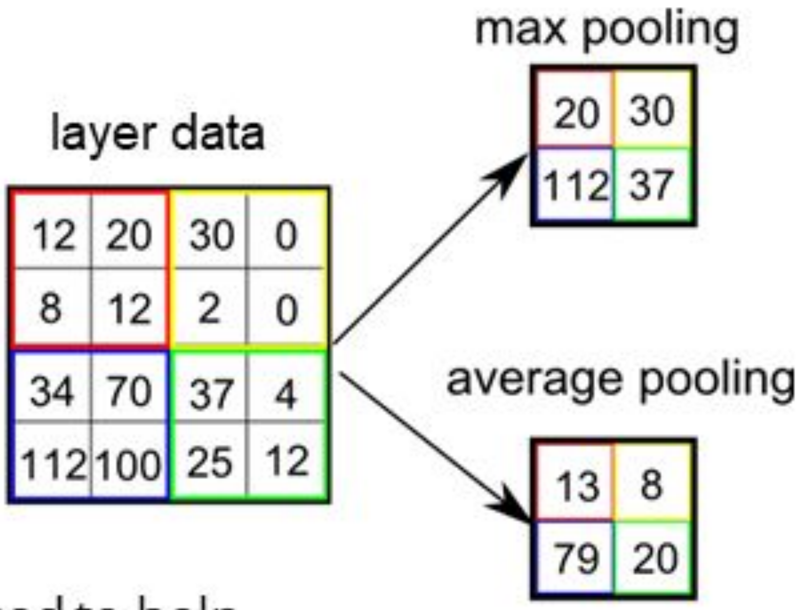
Pooling operation: Reduces size of the output and helps introduce invariances like translation in-sensitivity.

Common pooling operations:

(i) sub-sampling

(ii) averaging

(iii) max-pooling



source: Medium

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

Layer Operations

ReLU + bias

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

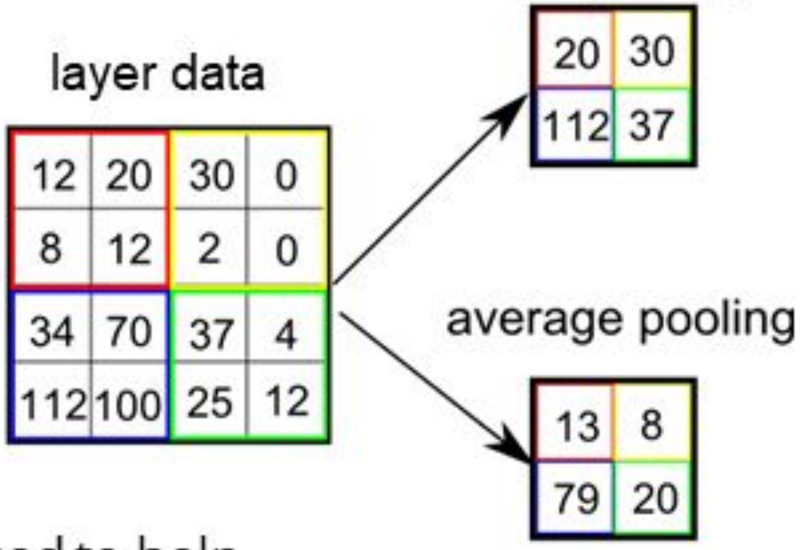
Pooling operation: Reduces size of the output and helps introduce invariances like translation in-sensitivity.

Common pooling operations:

(i) **sub-sampling**

(ii) **averaging**

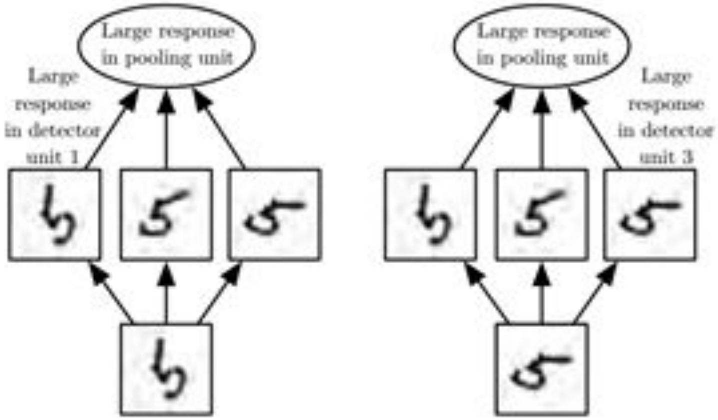
(iii) **max-pooling**



source: Medium

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

max pooling across channels

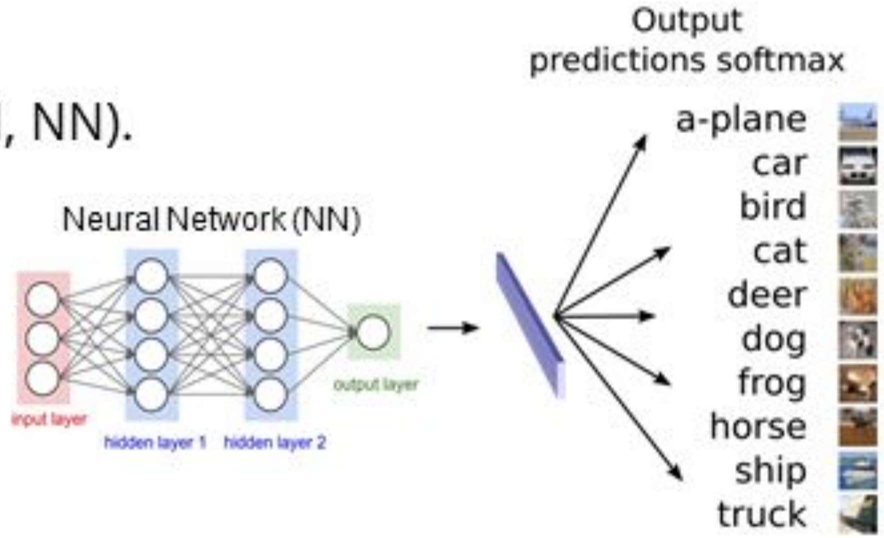


source: Goodfellow 2017

Layer Operations

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

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

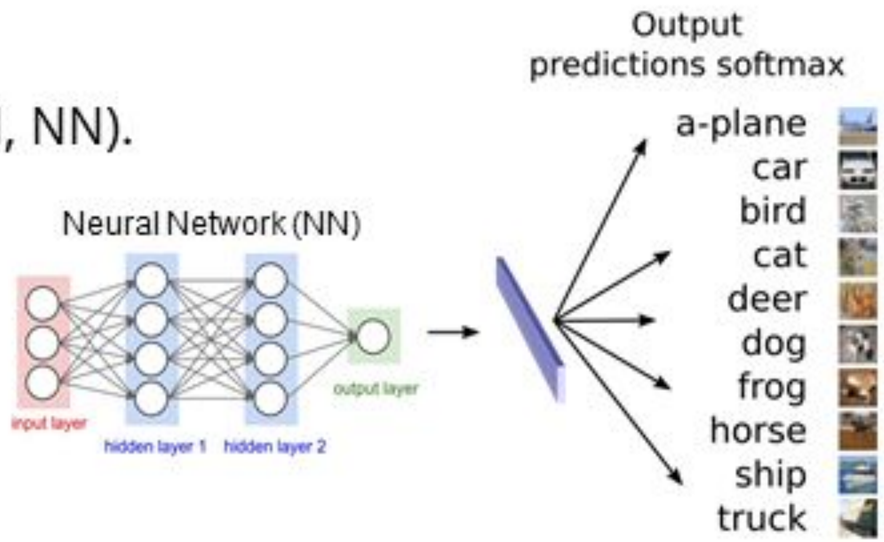


Layer Operations

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

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

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

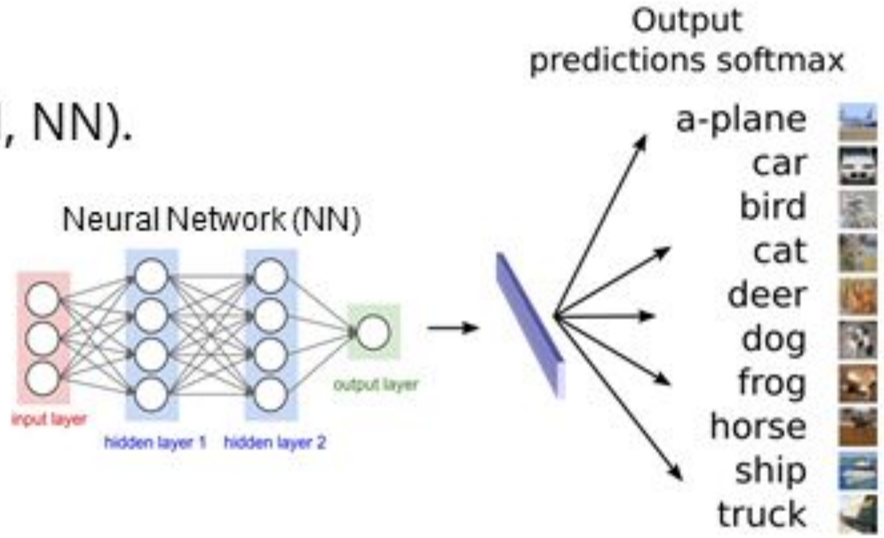


Layer Operations

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

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

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



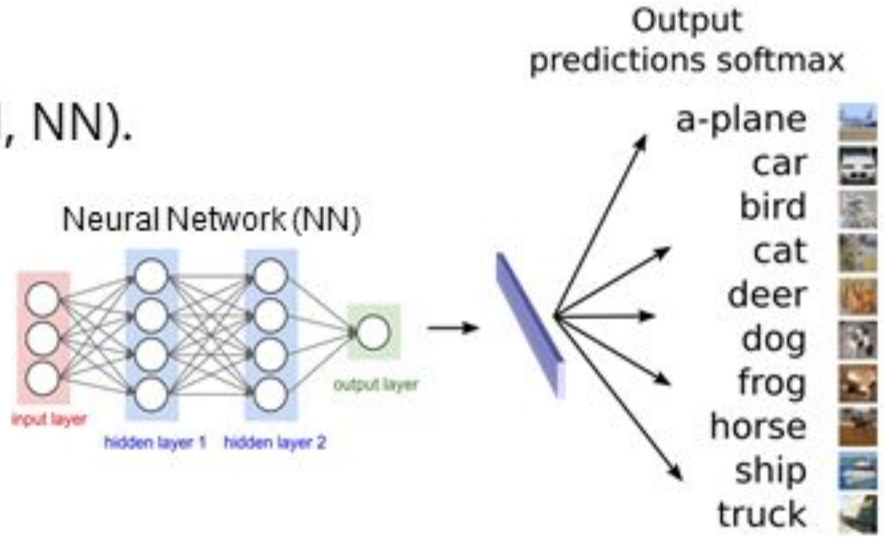
$1 - \text{hot vector}$

$$y = \left[\begin{array}{c}0\\1\\0\\0\\0\end{array}\right]\begin{array}{l}\\ \leftarrow k^{th}\text{ class}\\ \\ \\ \\ \end{array}$$

Layer Operations

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

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



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

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

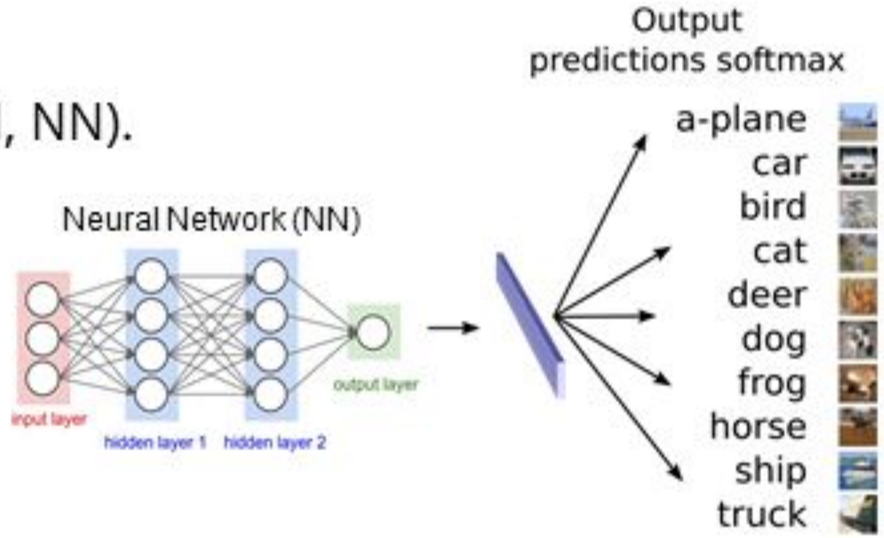
$1 - \text{hot vector}$

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow k^{th} \text{ class}$$

Layer Operations

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

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



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

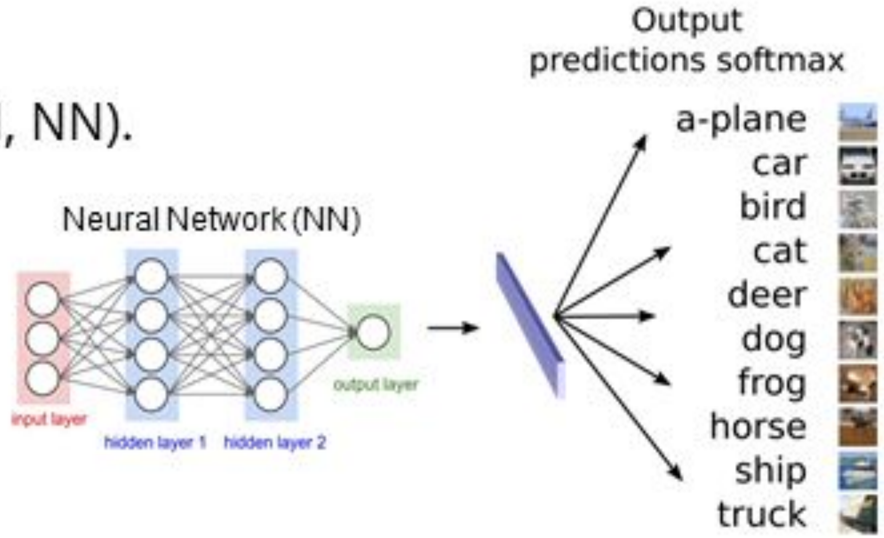
The output **q** ← **FFN** is turned into probability by softmax $\tilde{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10}\exp(q_i)} \to p(\tilde{y}|x)$.

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

$1 - \text{hot vector}$

$$\begin{array}{cc} y = \begin{bmatrix}0\\1\\0\\0\\0\end{bmatrix} & \begin{array}{l} \\ \leftarrow k^{th}\text{ class} \\ \\ \\ \\ \end{array} \end{array}$$

Layer Operations



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

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

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

The output **q** ← **FFN** is turned into probability by softmax $\tilde{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10}\exp(q_i)} \to p(\tilde{y}|x)$.

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

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

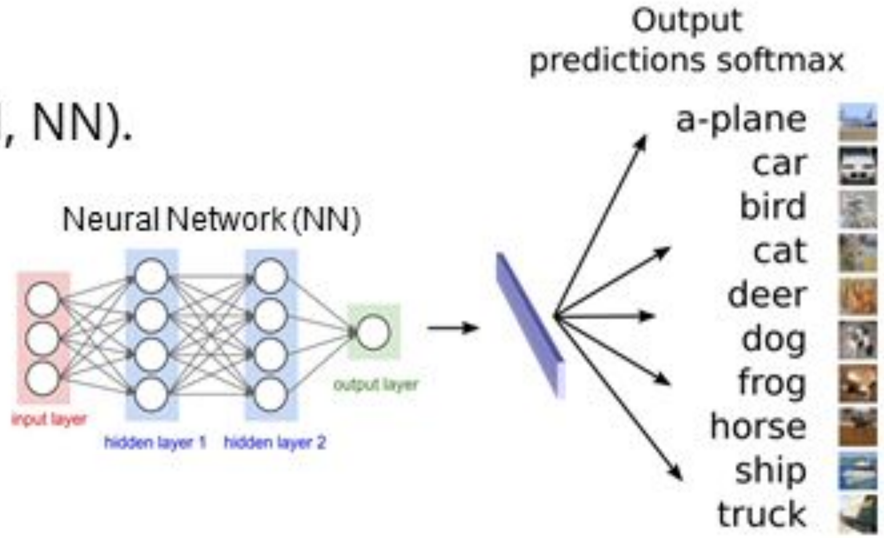
$1 - \text{hot vector}$

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow k^{th} \text{ class}$$

Layer Operations

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

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



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

The output **q** ← **FFN** is turned into probability by softmax $\tilde{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10}\exp(q_i)} \to p(\tilde{y}|x)$.

Training: back-propagation from loss $\mathrm{L}(\mathcal{S})$ through the FNN + conv layers.

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

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

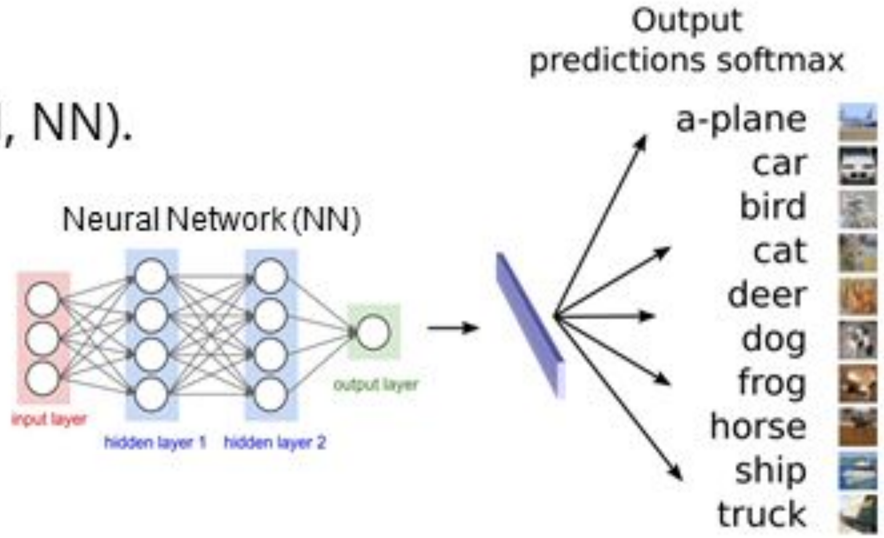
$1 - \text{hot vector}$

$$\mathrm{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} \\ \leftarrow k^{th} \text{ class} \\ \\ \\ \\ \end{array}$$

Layer Operations

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

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



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

The output **q** ← **FFN** is turned into probability by softmax $\tilde{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10}\exp(q_i)} \to p(\tilde{y}|x)$.

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

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

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

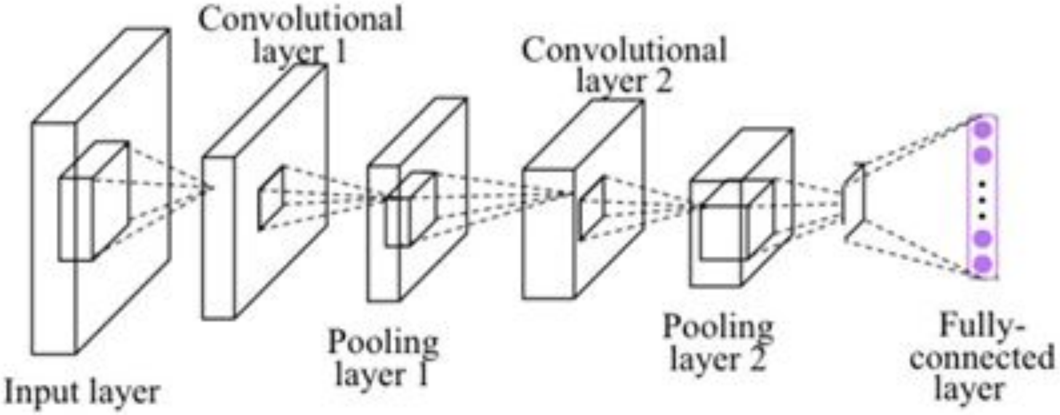
$1 - \text{hot vector}$

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \\ \leftarrow k^{th} \text{ class} \\ \\ \\ \\ \end{matrix}$$

Optimization only needs $\mathrm{L}(S)$ part, instead of full $\tilde{L}(S)$, to adjust the weights yielding $\tilde{y}$.

Convolutional Neural Network: Summary

Typical Architecture

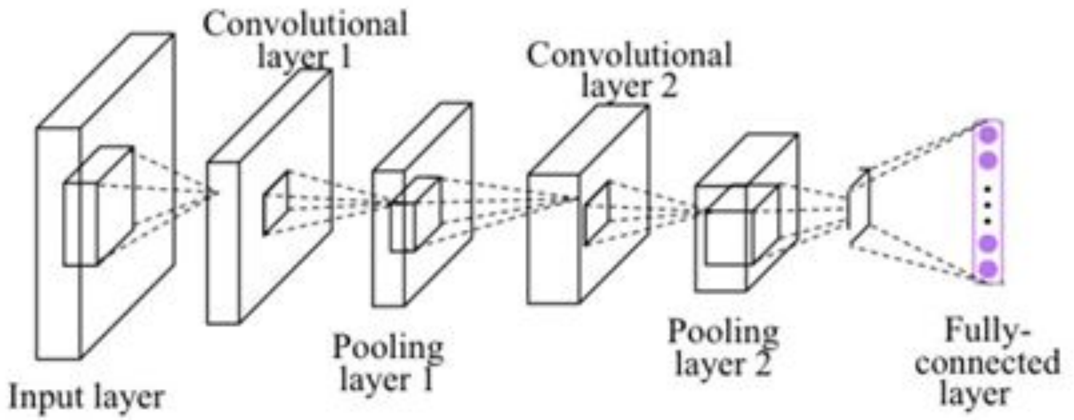


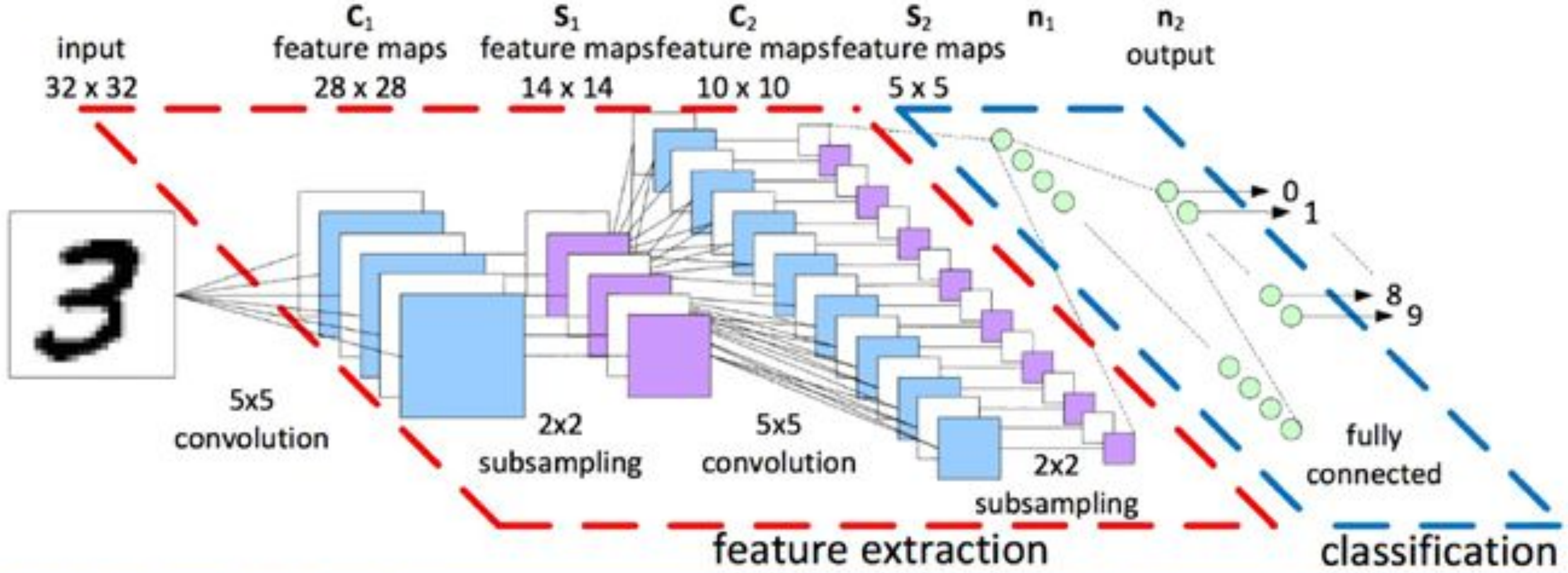
Input image is split into "channels" -> RGB.

Convolution layers extract features from the image into feature channels.

Convolutional Neural Network: Summary

Typical Architecture



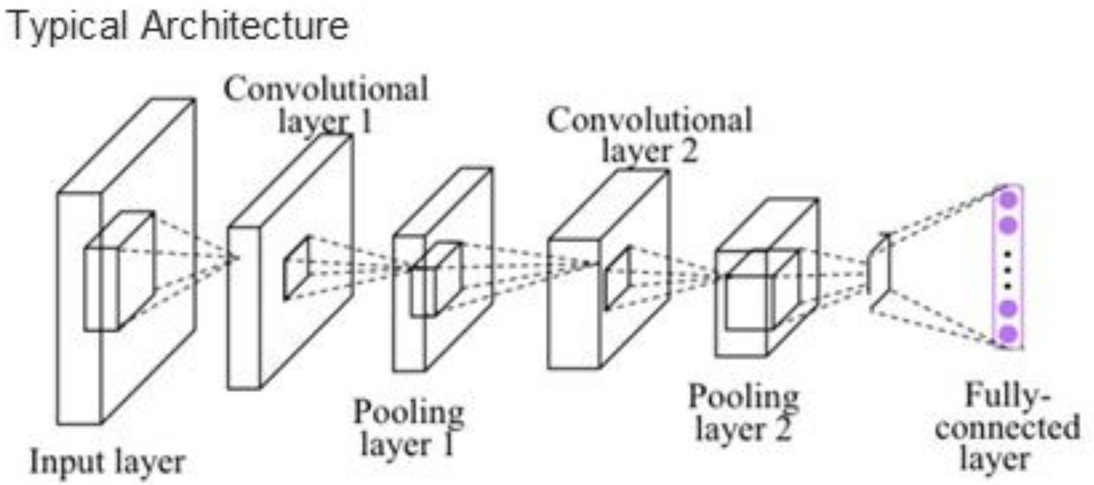


Input image is split into "channels" -> RGB.

Convolution layers extract features from the image into feature channels.

Source: Tutorial by Chongruo Wu

Convolutional Neural Network: Summary

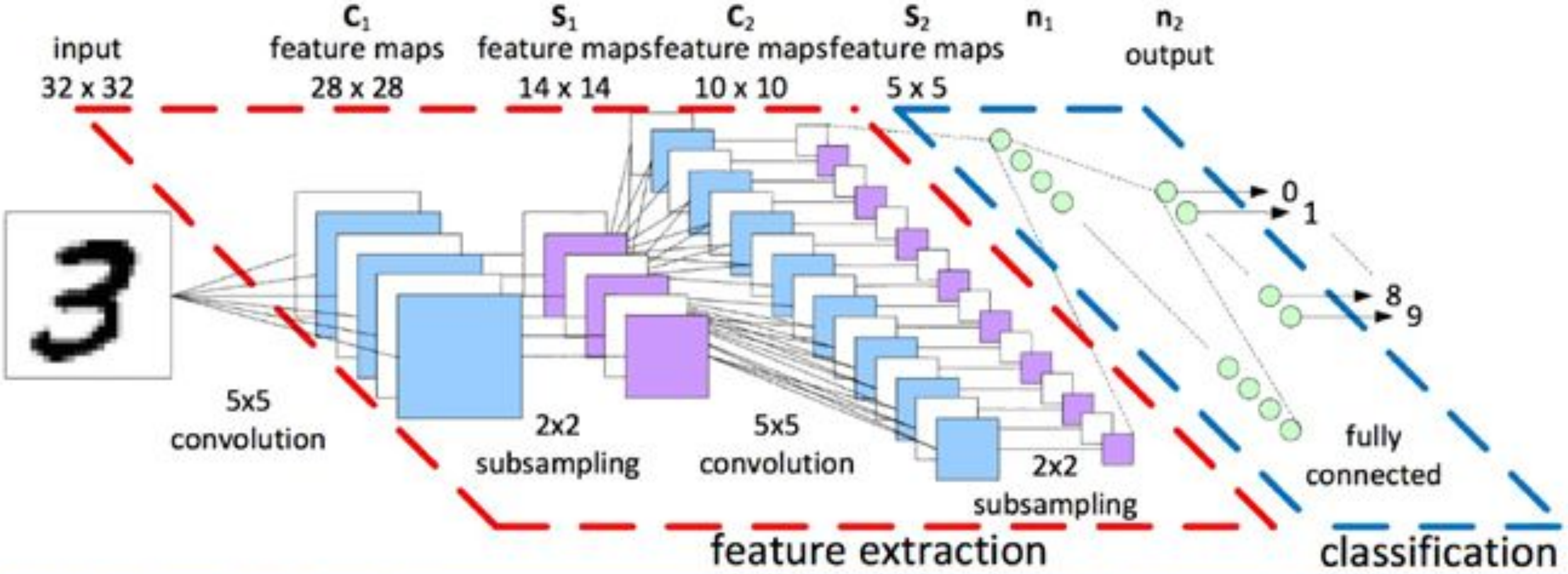


Input image is split into "channels" -> RGB.

Convolution layers extract features from the image into feature channels.

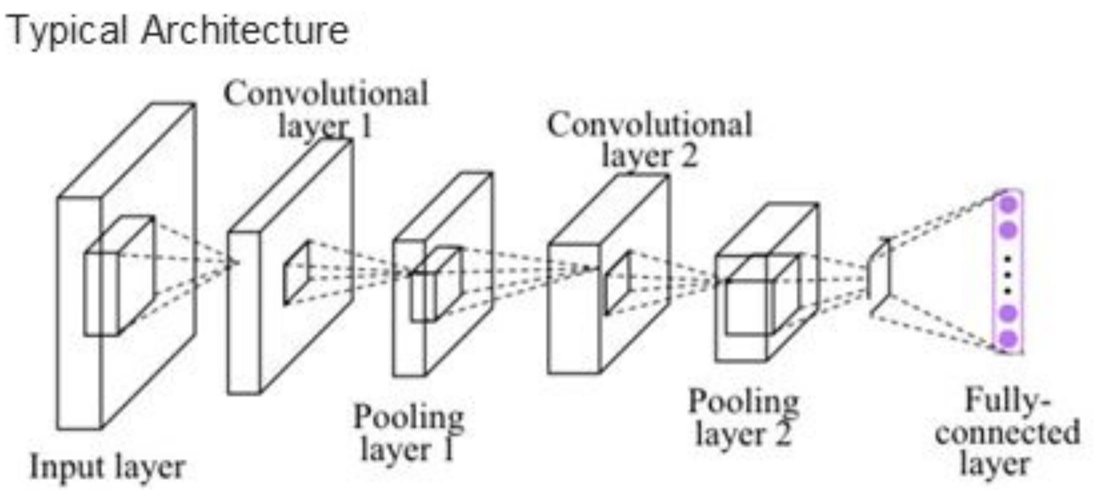
Detector operation typically ReLU + bias.

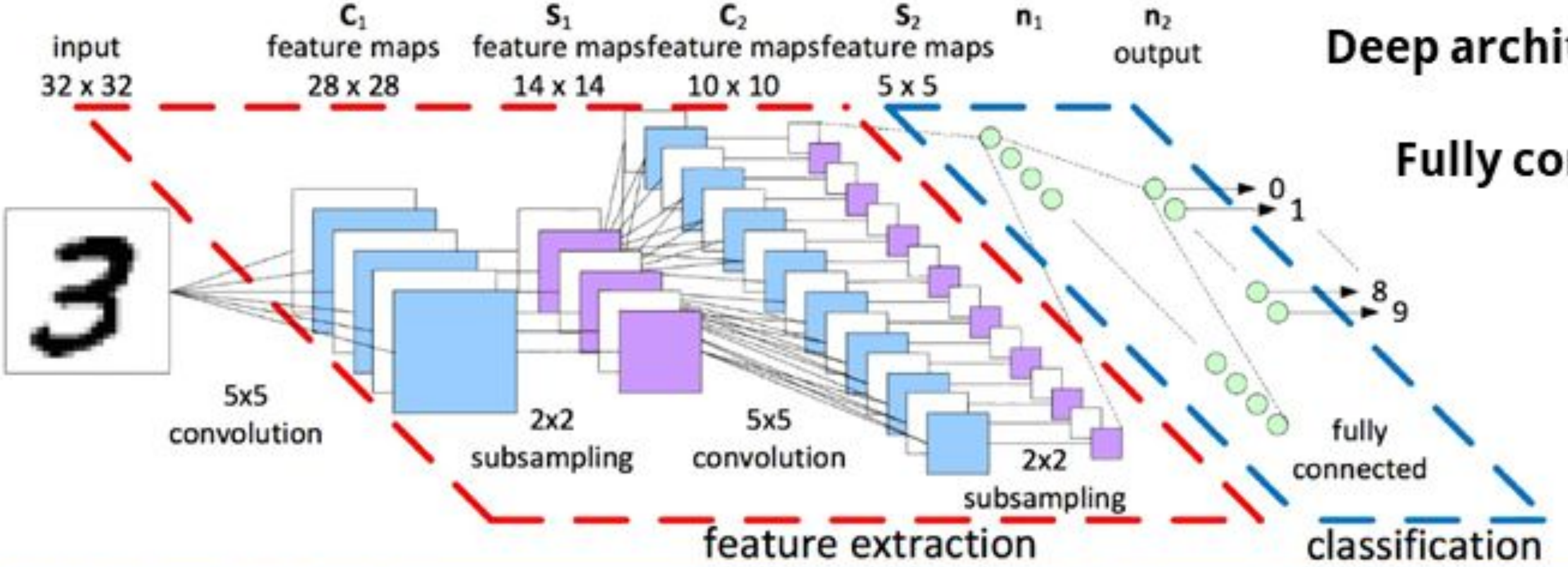
Pooling reduces entries in channel (enhances invariance to image transforms)



Source: Tutorial by Chongruo Wu

Convolutional Neural Network: Summary





Input image is split into "channels" -> RGB.

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)

Deep architectures stack to repeat.

Fully connected layer → predicts class.

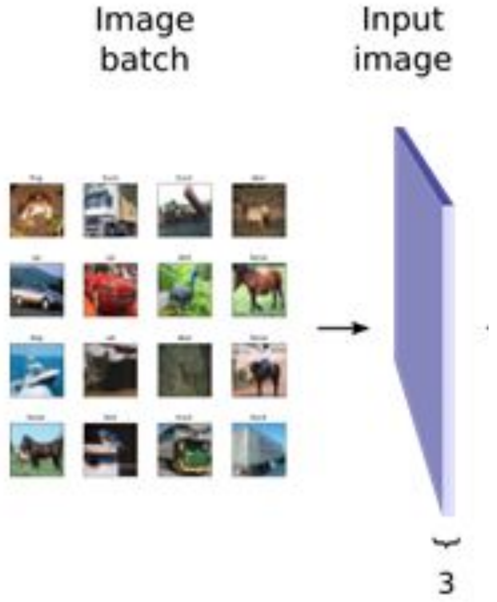
Source: Tutorial by Chongruo Wu

CIFAR10: Image Classification

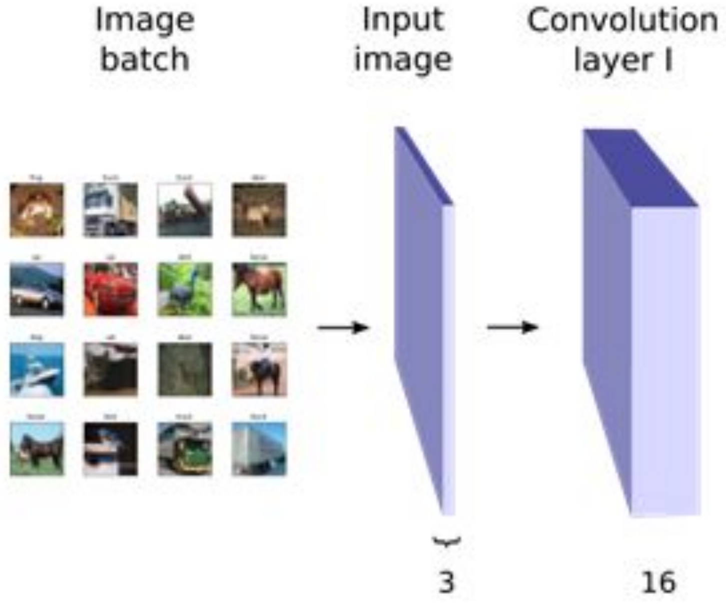
CIFAR-10: Convolutional Neural Network

Image
batch

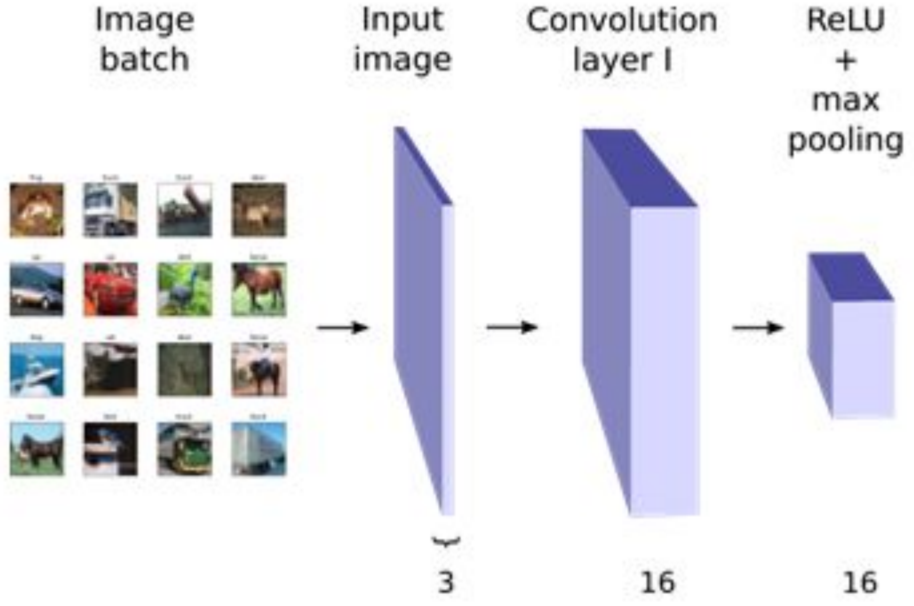
CIFAR-10: Convolutional Neural Network



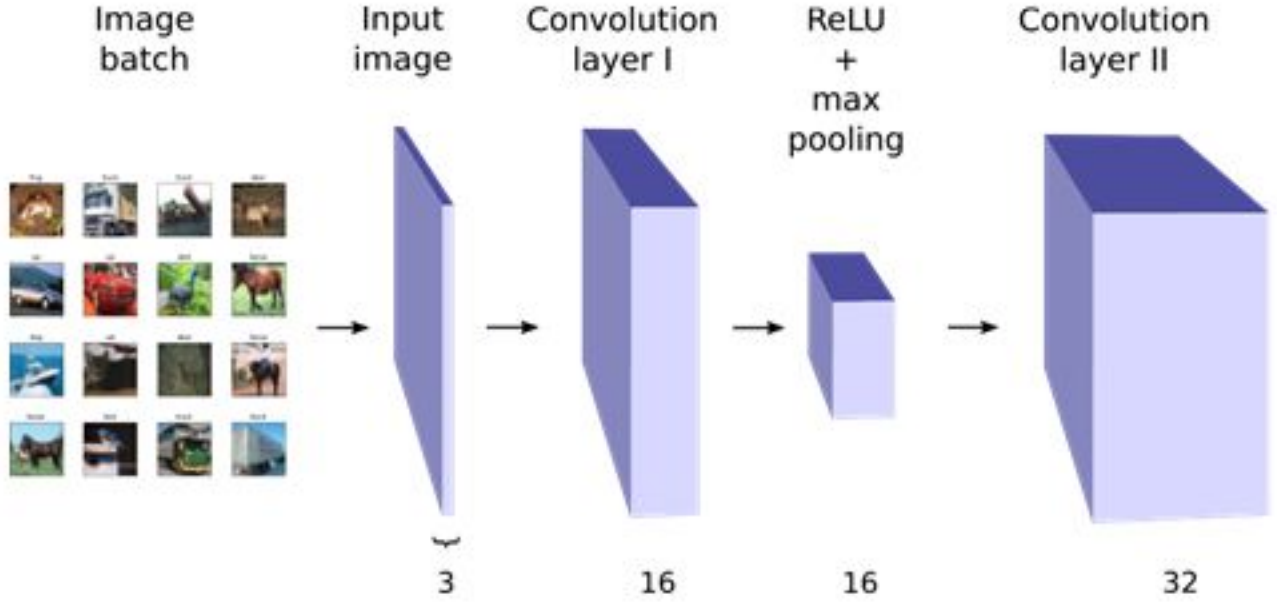
CIFAR-10: Convolutional Neural Network



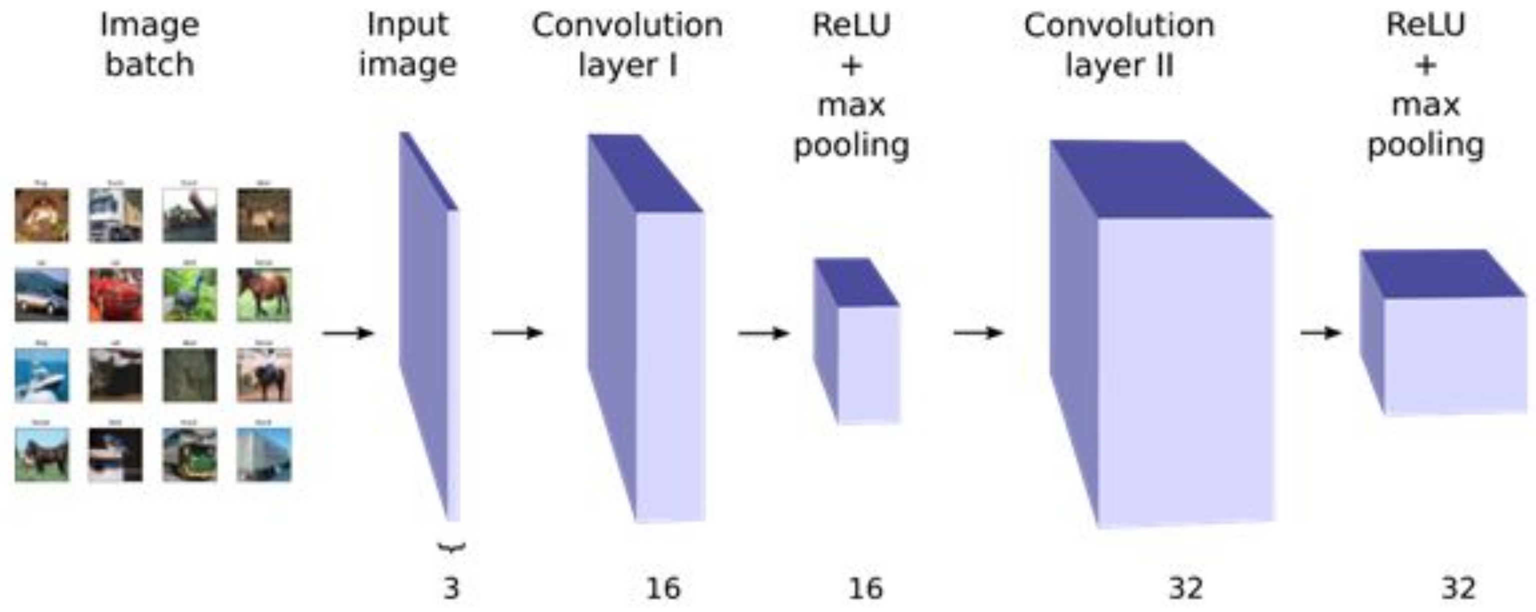
CIFAR-10: Convolutional Neural Network



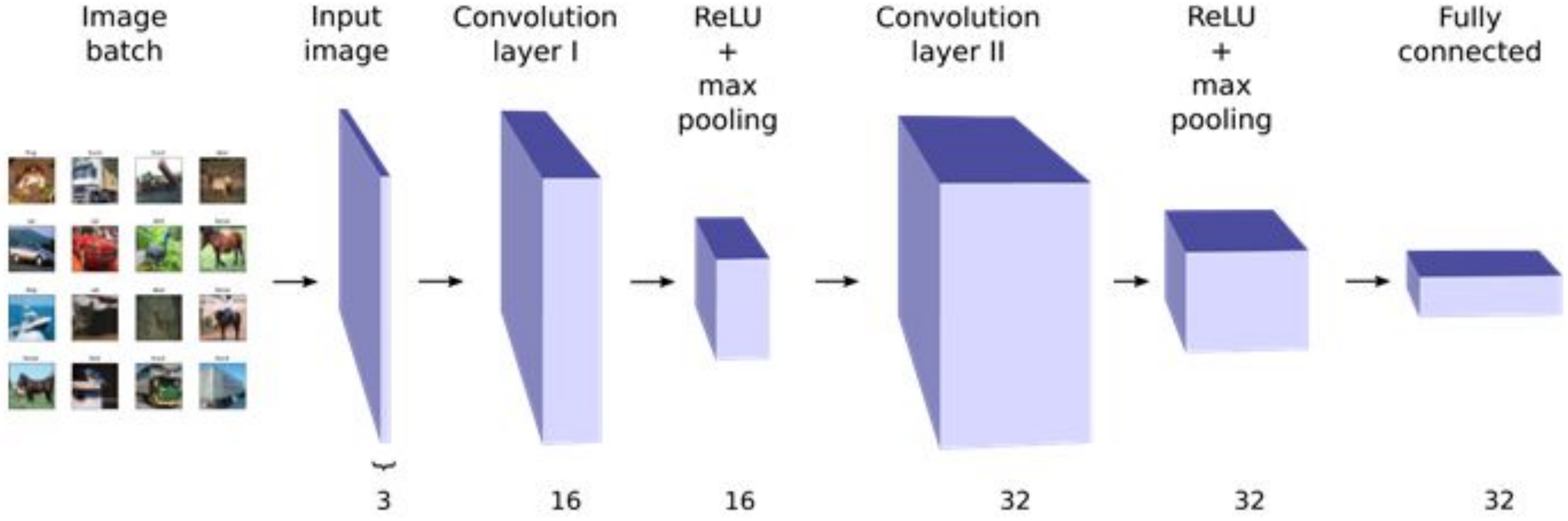
CIFAR-10: Convolutional Neural Network



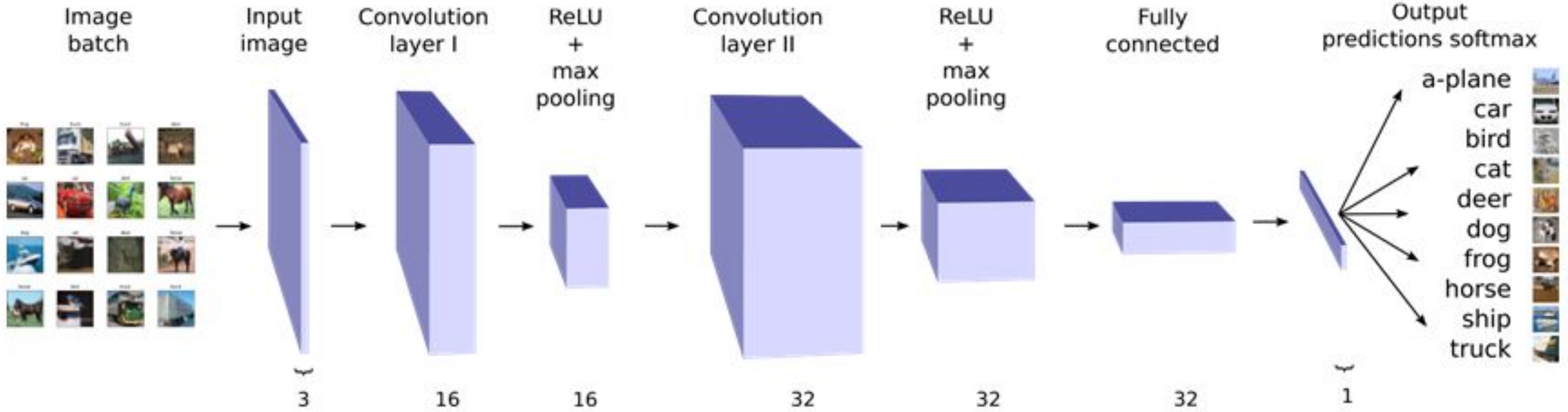
CIFAR-10: Convolutional Neural Network



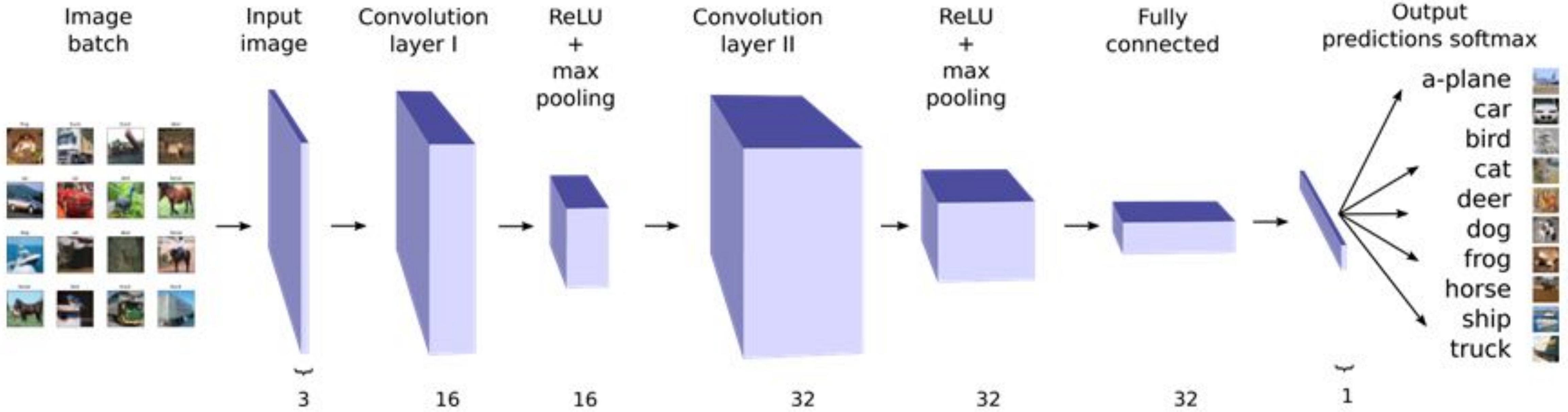
CIFAR-10: Convolutional Neural Network



CIFAR-10: Convolutional Neural Network

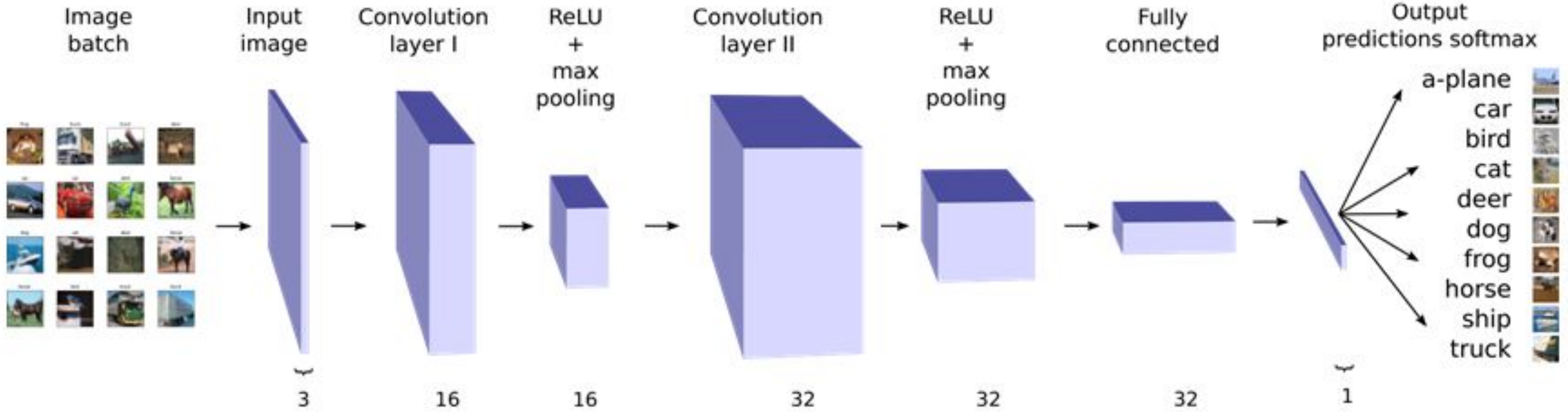


CIFAR-10: Convolutional Neural Network



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).

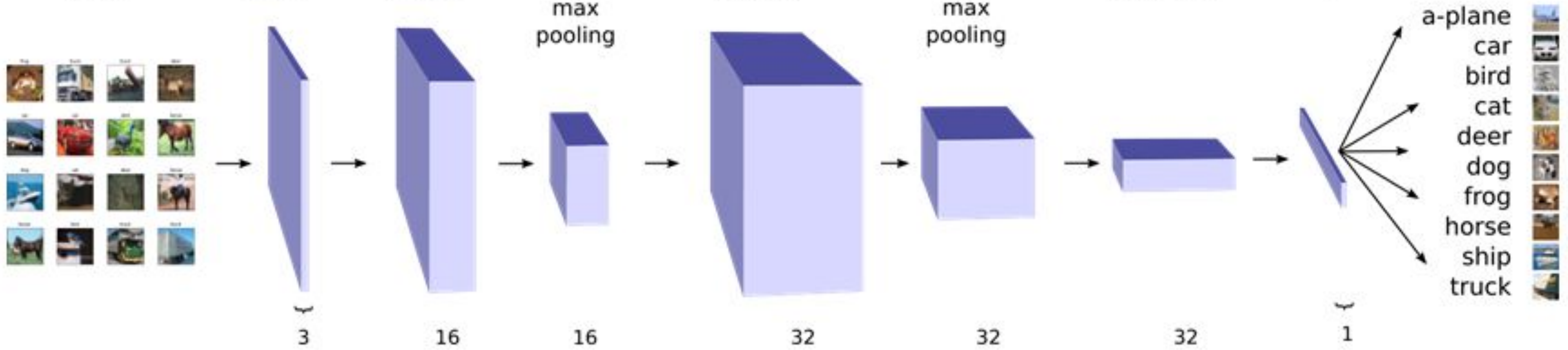
CIFAR-10: Convolutional Neural Network



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).

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

CIFAR-10: Convolutional Neural Network



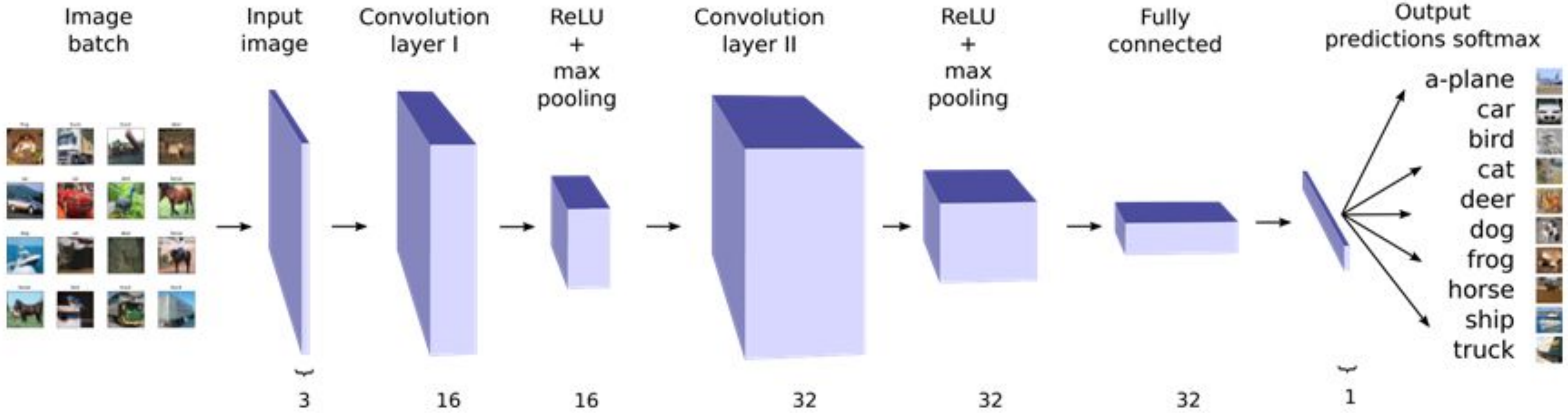
CIFAR-10 Dataset: consists of 60,000 images 32x32 pixels RGB with labels in 10 categories (plane, car, bird, cat, deer, dog, frog, horse, ship, truck).

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

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

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow i^{th} \text{ class}$$

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).

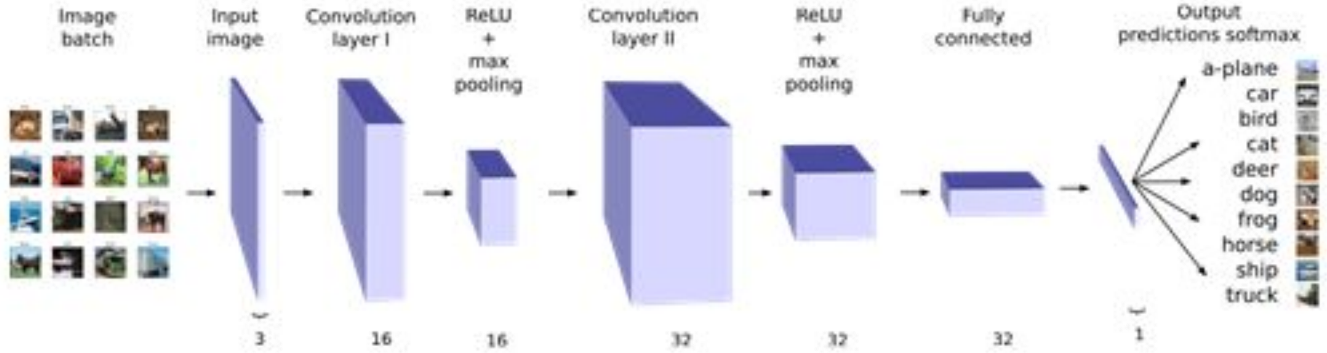
2 convolution layers + fully connected layer → q_i predicts probability of class via softmax: $\tilde{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)}$.

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

Optimization: Stochastic Gradient Descent with batch size 100 images.

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{array}{l} \\ \leftarrow i^{th} \text{ class} \\ \\ \\ \\ \end{array}$$

CIFAR-10: Convolutional Neural Network



Training:

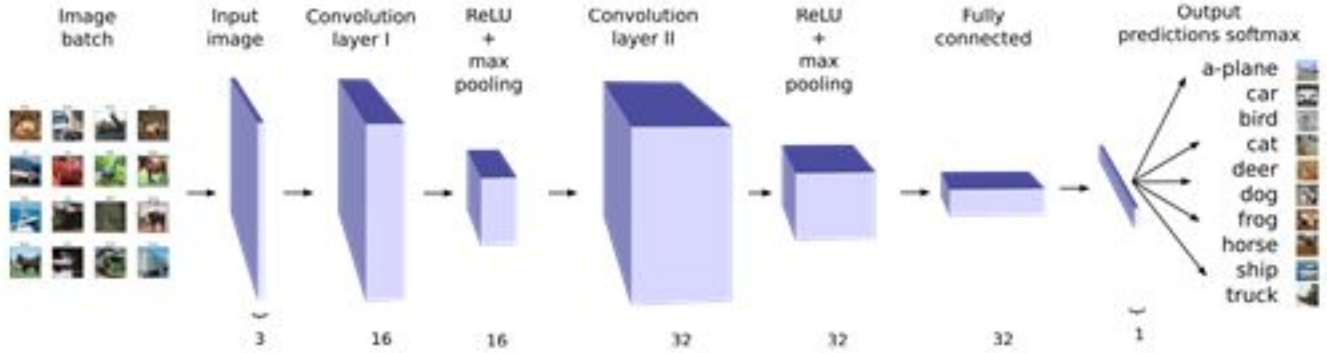
Cross-entropy loss: $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} -y_j^{(i)} \log(\tilde{y}_j^{(i)})$.

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: $\mathrm{L}(S) = \sum_{i=1}^{m}\sum_{j=1}^{10} - y_j^{(i)}\log(\tilde{y}_j^{(i)})$.

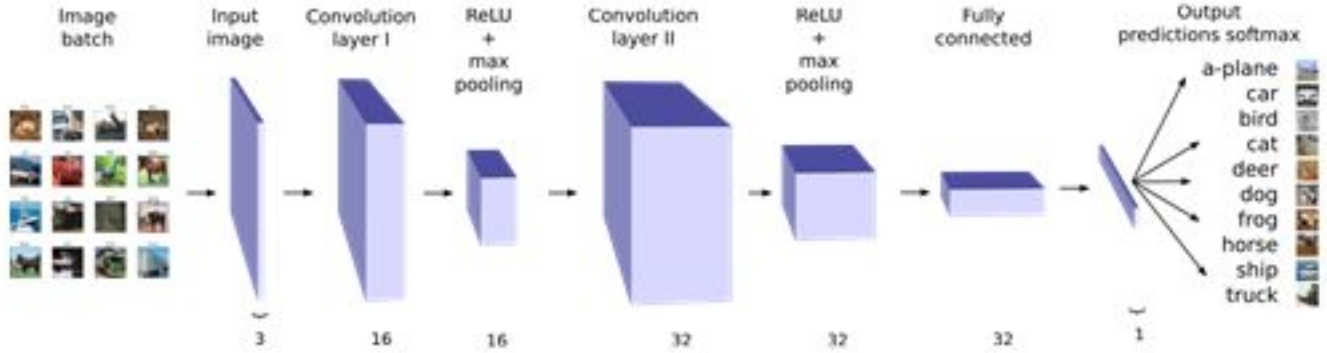
Random initial weights.

Mini-batches of size 100.

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

```
Training the CNN with:
numEpochs = 5
batchSize = 100

Epoch: [1/5]; batchStep = [100/500]; Loss: 1.4402.
Epoch: [1/5]; batchStep = [200/500]; Loss: 1.2206.
Epoch: [1/5]; batchStep = [300/500]; Loss: 1.2734.
Epoch: [1/5]; batchStep = [400/500]; Loss: 1.1563.
Epoch: [1/5]; batchStep = [500/500]; Loss: 1.1195.
Epoch: [2/5]; batchStep = [100/500]; Loss: 0.9954.
Epoch: [2/5]; batchStep = [200/500]; Loss: 1.1832.
Epoch: [2/5]; batchStep = [300/500]; Loss: 1.2185.
Epoch: [2/5]; batchStep = [400/500]; Loss: 0.9191.
Epoch: [2/5]; batchStep = [500/500]; Loss: 0.9215.
Epoch: [3/5]; batchStep = [100/500]; Loss: 1.0034.
Epoch: [3/5]; batchStep = [200/500]; Loss: 0.9616.
Epoch: [3/5]; batchStep = [300/500]; Loss: 0.8275.
Epoch: [3/5]; batchStep = [400/500]; Loss: 0.9199.
Epoch: [3/5]; batchStep = [500/500]; Loss: 0.9362.
Epoch: [4/5]; batchStep = [100/500]; Loss: 0.7794.
Epoch: [4/5]; batchStep = [200/500]; Loss: 0.9632.
Epoch: [4/5]; batchStep = [300/500]; Loss: 0.7169.
Epoch: [4/5]; batchStep = [400/500]; Loss: 0.6411.
Epoch: [4/5]; batchStep = [500/500]; Loss: 0.8790.
Epoch: [5/5]; batchStep = [100/500]; Loss: 0.7935.
Epoch: [5/5]; batchStep = [200/500]; Loss: 0.8741.
Epoch: [5/5]; batchStep = [300/500]; Loss: 0.7293.
Epoch: [5/5]; batchStep = [400/500]; Loss: 0.7524.
Epoch: [5/5]; batchStep = [500/500]; Loss: 0.7942.
```

CIFAR-10: Convolutional Neural Network




```
Training the CNN with:
numEpochs = 5
batchSize = 100

Epoch: [1/5]; batchStep = [100/500]; Loss: 1.4402.
Epoch: [1/5]; batchStep = [200/500]; Loss: 1.2206.
Epoch: [1/5]; batchStep = [300/500]; Loss: 1.2734.
Epoch: [1/5]; batchStep = [400/500]; Loss: 1.1563.
Epoch: [1/5]; batchStep = [500/500]; Loss: 1.1195.
Epoch: [2/5]; batchStep = [100/500]; Loss: 0.9954.
Epoch: [2/5]; batchStep = [200/500]; Loss: 1.1832.
Epoch: [2/5]; batchStep = [300/500]; Loss: 1.2185.
Epoch: [2/5]; batchStep = [400/500]; Loss: 0.9191.
Epoch: [2/5]; batchStep = [500/500]; Loss: 0.9215.
Epoch: [3/5]; batchStep = [100/500]; Loss: 1.0034.
Epoch: [3/5]; batchStep = [200/500]; Loss: 0.9616.
Epoch: [3/5]; batchStep = [300/500]; Loss: 0.8275.
Epoch: [3/5]; batchStep = [400/500]; Loss: 0.9199.
Epoch: [3/5]; batchStep = [500/500]; Loss: 0.9362.
Epoch: [4/5]; batchStep = [100/500]; Loss: 0.7794.
Epoch: [4/5]; batchStep = [200/500]; Loss: 0.9632.
Epoch: [4/5]; batchStep = [300/500]; Loss: 0.7169.
Epoch: [4/5]; batchStep = [400/500]; Loss: 0.6411.
Epoch: [4/5]; batchStep = [500/500]; Loss: 0.8790.
Epoch: [5/5]; batchStep = [100/500]; Loss: 0.7935.
Epoch: [5/5]; batchStep = [200/500]; Loss: 0.8741.
Epoch: [5/5]; batchStep = [300/500]; Loss: 0.7293.
Epoch: [5/5]; batchStep = [400/500]; Loss: 0.7524.
Epoch: [5/5]; batchStep = [500/500]; Loss: 0.7942.
```

Training:

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

Random initial weights.

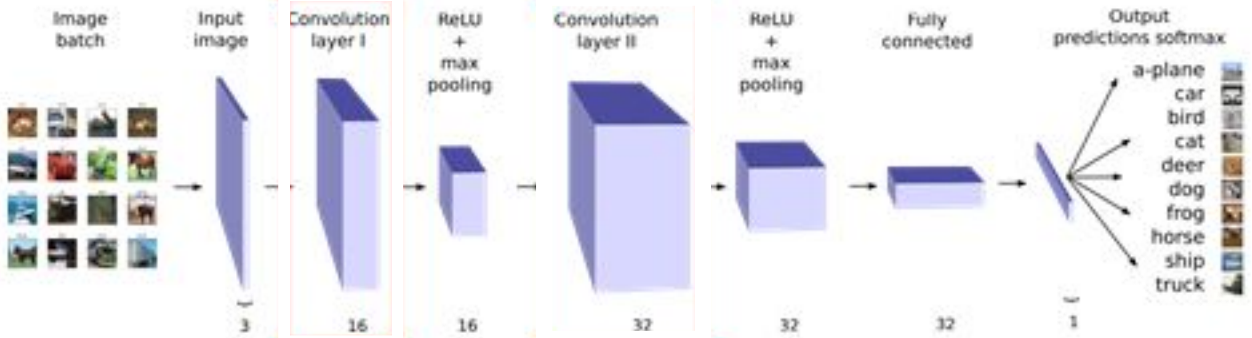
Mini-batches of size 100.

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

Final training loss of $7.8x10^{-1}$.

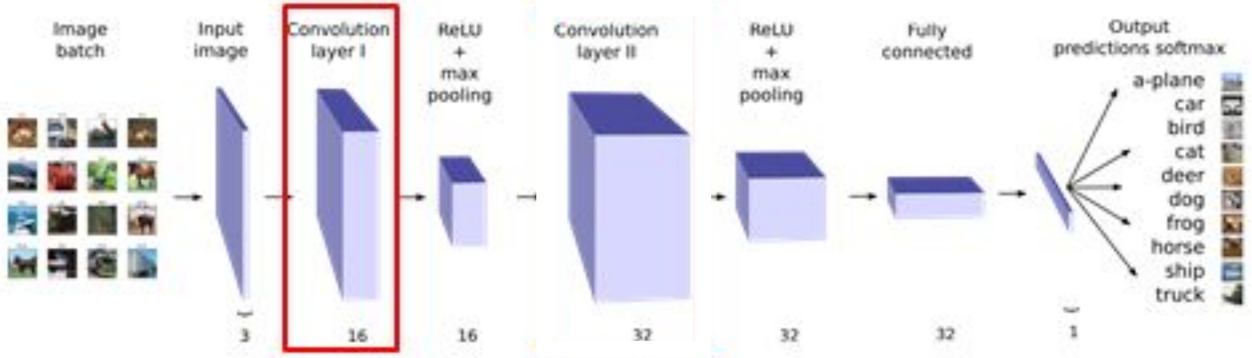
CIFAR-10: Convolutional Neural Network

Hidden Layers



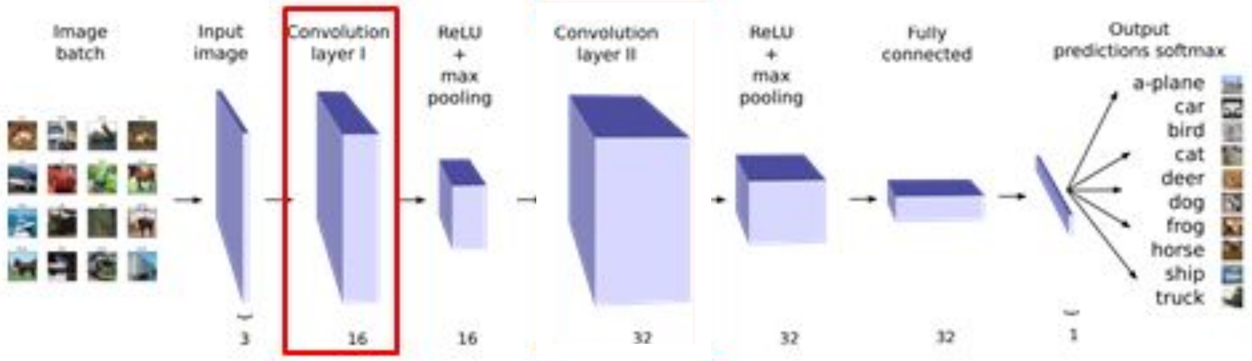
CIFAR-10: Convolutional Neural Network

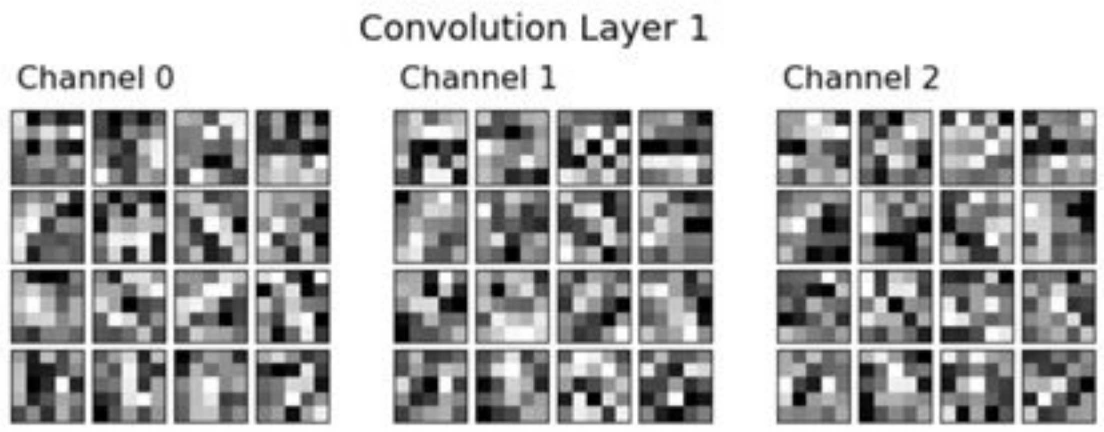
Hidden Layers



CIFAR-10: Convolutional Neural Network

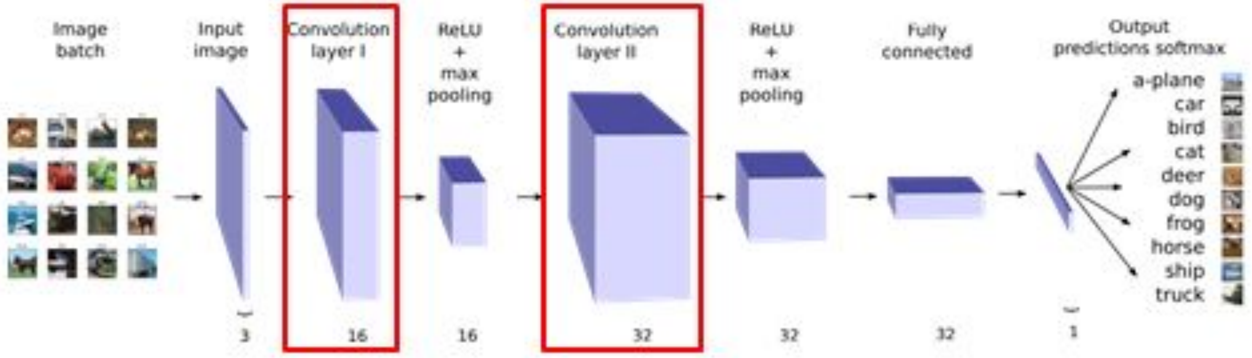
Hidden Layers

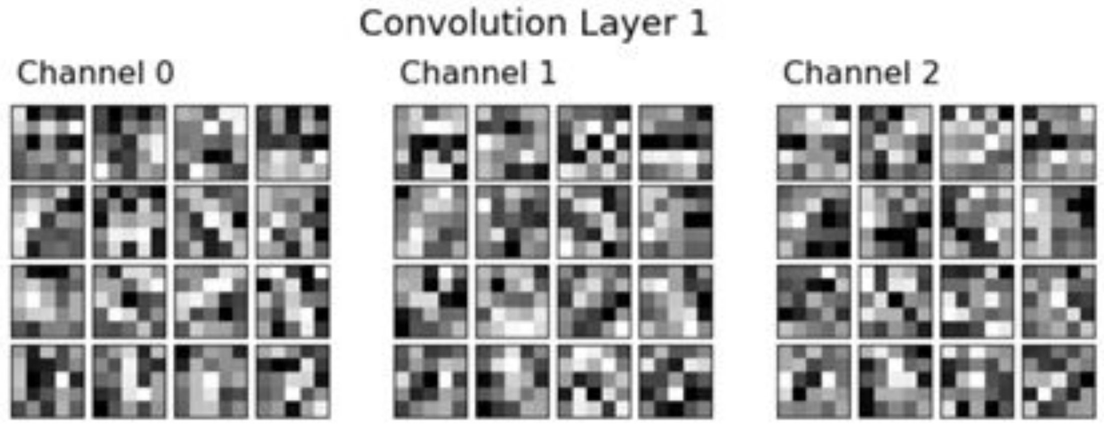




CIFAR-10: Convolutional Neural Network

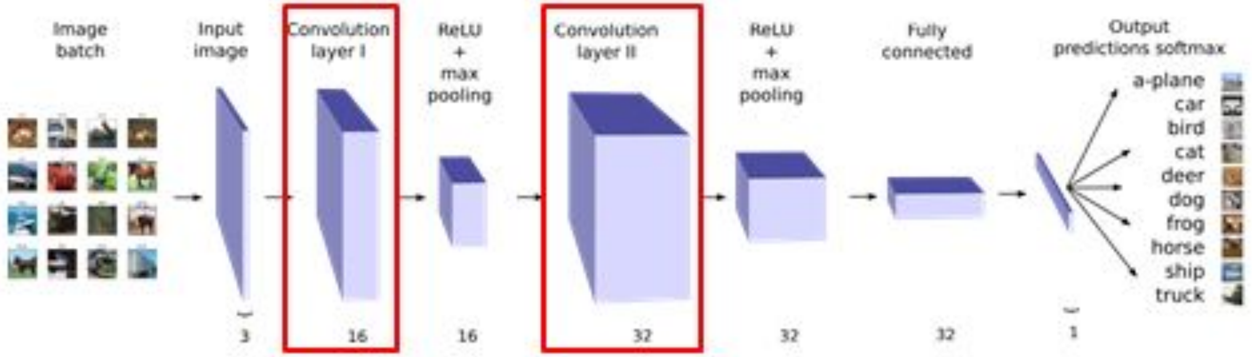
Hidden Layers

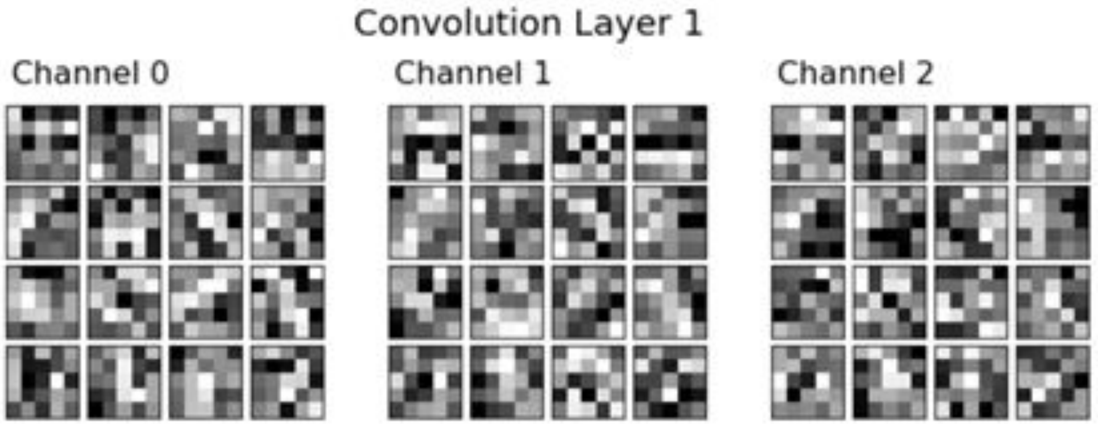


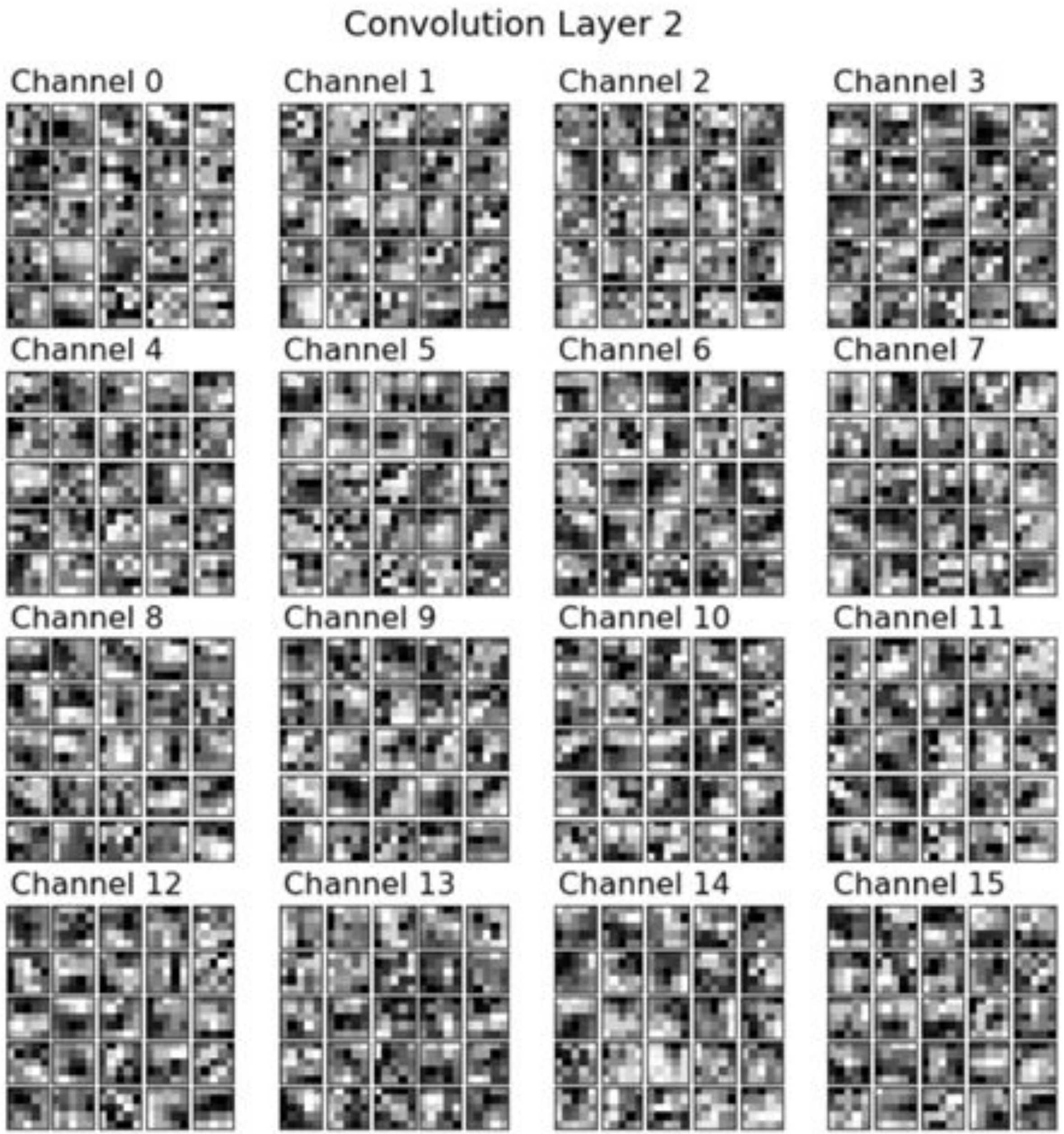


CIFAR-10: Convolutional Neural Network

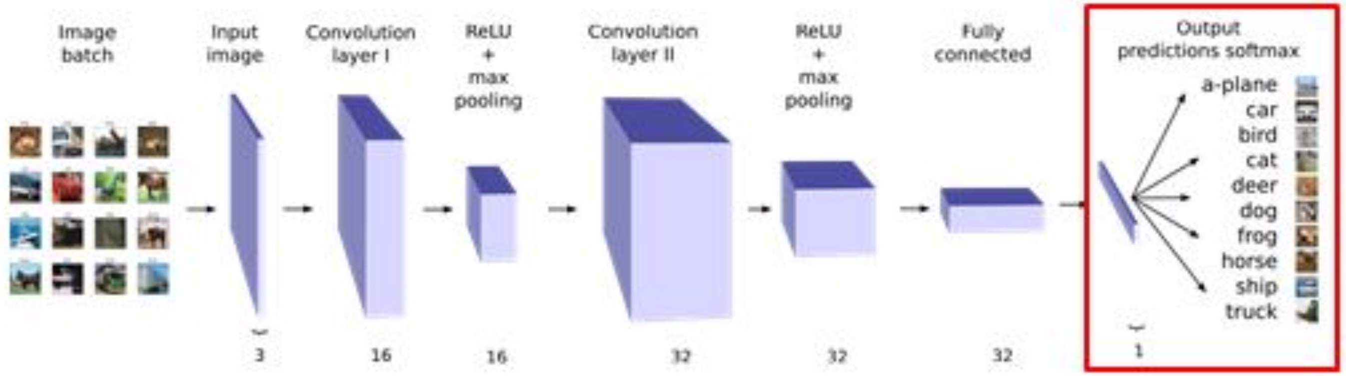
Hidden Layers







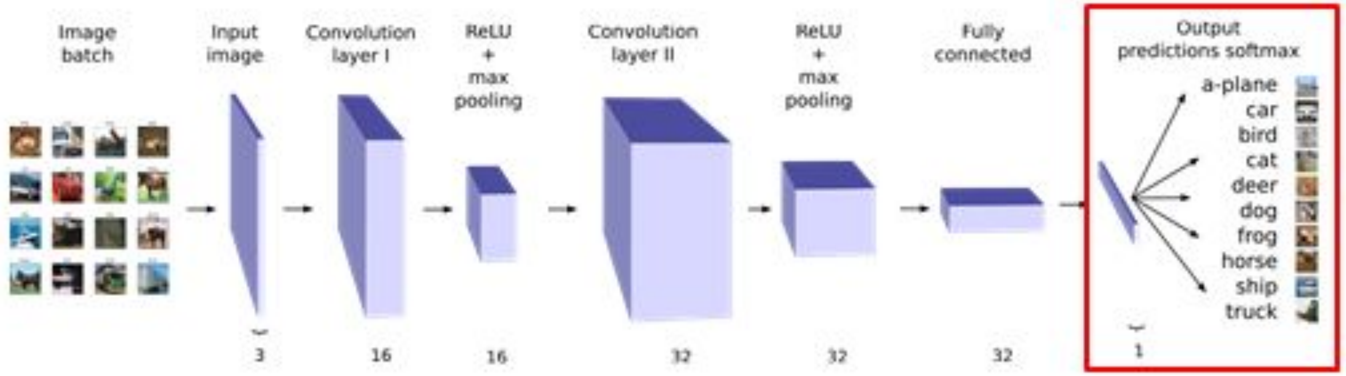
CIFAR-10: Convolutional Neural Network



Results:

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

CIFAR-10: Convolutional Neural Network

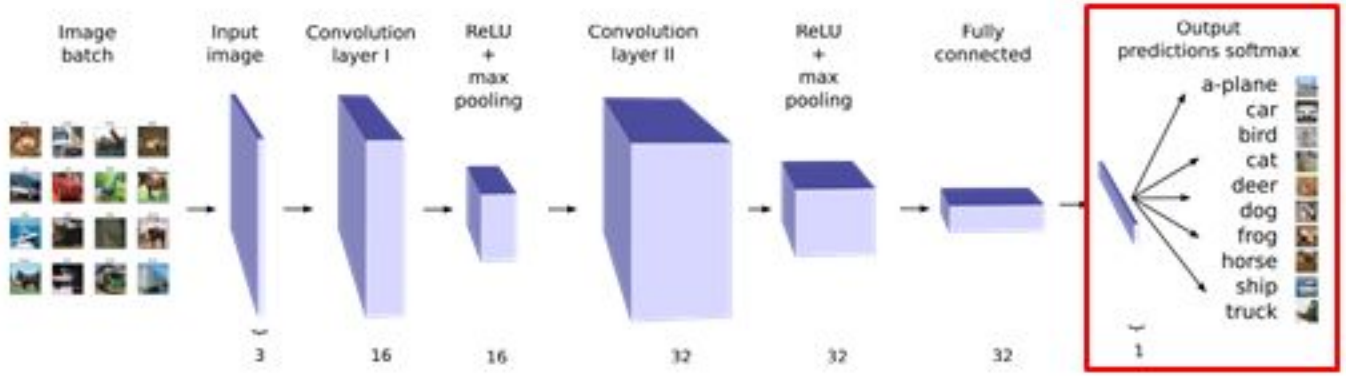


Results:

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

```
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 accurary of 69.39%
```

CIFAR-10: Convolutional Neural Network

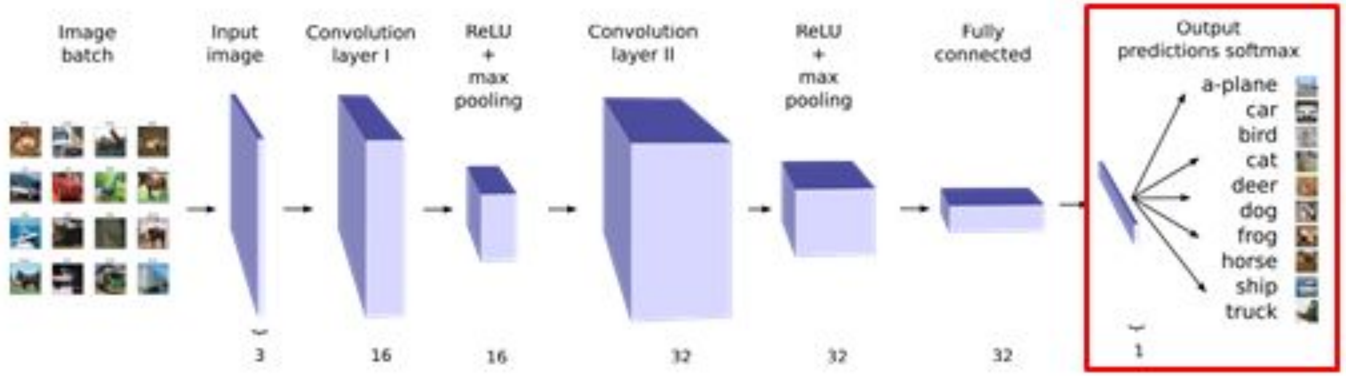


Results:

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

```
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 accurary of 69.39%
```

CIFAR-10: Convolutional Neural Network



Results:

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

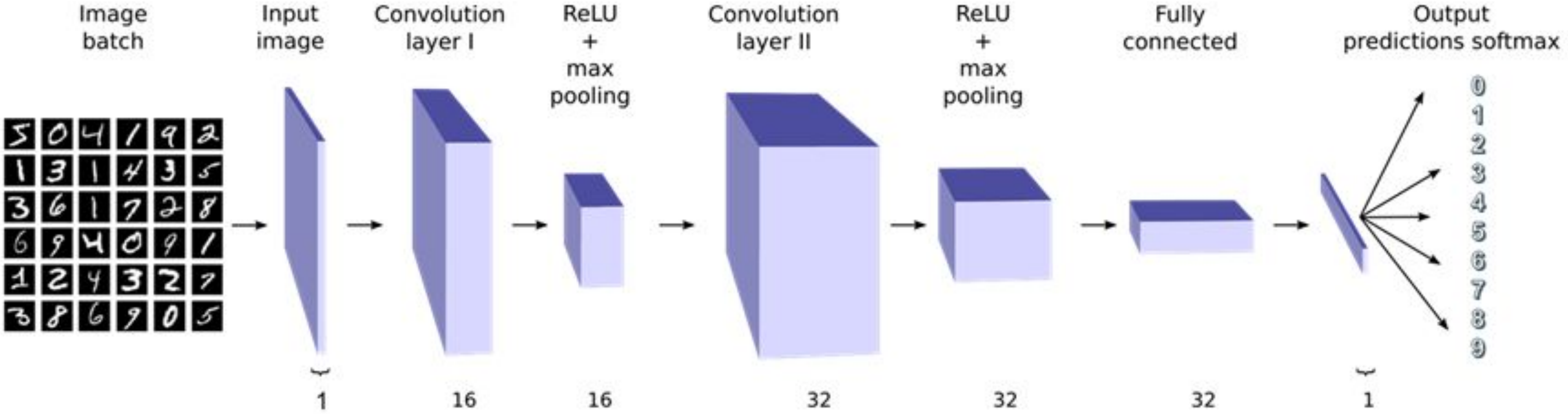
Achieves a test accuracy of 69%.

This could be improved by further adjustments of hyper-parameters, tuning of architecture, additional layers, training protocols, among other strategies.

```
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 accurary of 69.39%
```

MNIST: Digit Classification

MNIST: Convolutional Neural Network



MNIST Dataset: consists of 60,000 images 28x28 pixels grayscale with labels in 10 categories (digits).

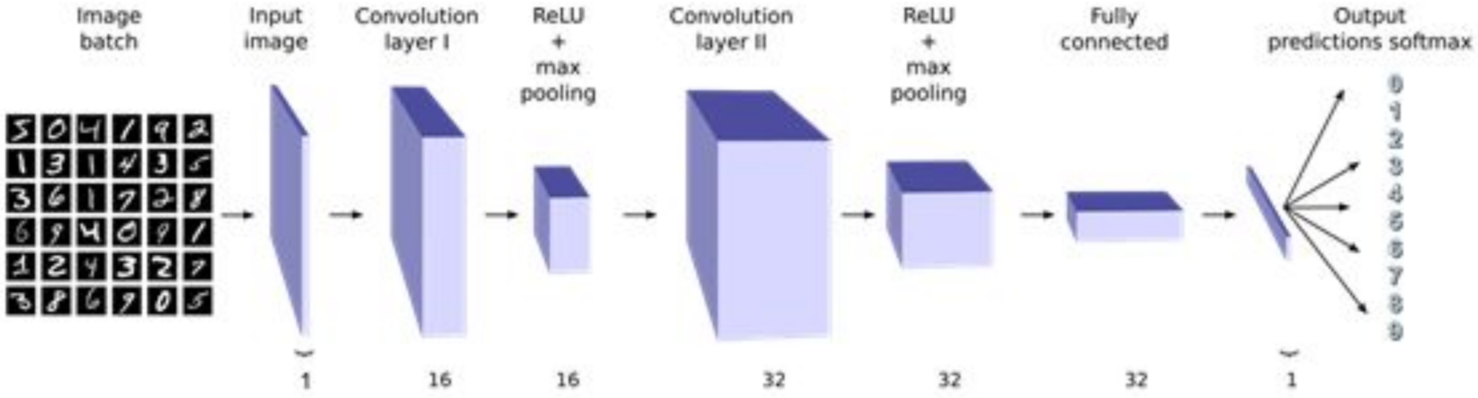
2 convolution layers + fully connected layer → q_i predicts probability of class via softmax: $\tilde{y}_i = \frac{\exp(q_i)}{\sum_{i=1}^{10} \exp(q_i)}$.

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

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \leftarrow i^{th} \text{ class}$$

Optimization: Stochastic Gradient Descent with batch size 100 images.

MNIST: Convolutional Neural Network




```
Training the CNN with:
numEpochs = 5
batchSize = 100

Epoch: [1/5]; batchStep = [100/600]; Loss: 0.1506.
Epoch: [1/5]; batchStep = [200/600]; Loss: 0.0758.
Epoch: [1/5]; batchStep = [300/600]; Loss: 0.1095.
Epoch: [1/5]; batchStep = [400/600]; Loss: 0.0388.
Epoch: [1/5]; batchStep = [500/600]; Loss: 0.0507.
Epoch: [1/5]; batchStep = [600/600]; Loss: 0.1308.
Epoch: [2/5]; batchStep = [100/600]; Loss: 0.0737.
Epoch: [2/5]; batchStep = [200/600]; Loss: 0.0230.
Epoch: [2/5]; batchStep = [300/600]; Loss: 0.0317.
Epoch: [2/5]; batchStep = [400/600]; Loss: 0.0785.
Epoch: [2/5]; batchStep = [500/600]; Loss: 0.0212.
Epoch: [2/5]; batchStep = [600/600]; Loss: 0.0645.
Epoch: [3/5]; batchStep = [100/600]; Loss: 0.0196.
Epoch: [3/5]; batchStep = [200/600]; Loss: 0.1008.
Epoch: [3/5]; batchStep = [300/600]; Loss: 0.0190.
Epoch: [3/5]; batchStep = [400/600]; Loss: 0.0265.
Epoch: [3/5]; batchStep = [500/600]; Loss: 0.0208.
Epoch: [3/5]; batchStep = [600/600]; Loss: 0.0146.
Epoch: [4/5]; batchStep = [100/600]; Loss: 0.0591.
Epoch: [4/5]; batchStep = [200/600]; Loss: 0.0080.
Epoch: [4/5]; batchStep = [300/600]; Loss: 0.0132.
Epoch: [4/5]; batchStep = [400/600]; Loss: 0.0725.
Epoch: [4/5]; batchStep = [500/600]; Loss: 0.0169.
Epoch: [4/5]; batchStep = [600/600]; Loss: 0.0063.
Epoch: [5/5]; batchStep = [100/600]; Loss: 0.0055.
Epoch: [5/5]; batchStep = [200/600]; Loss: 0.0065.
Epoch: [5/5]; batchStep = [300/600]; Loss: 0.0647.
Epoch: [5/5]; batchStep = [400/600]; Loss: 0.0081.
Epoch: [5/5]; batchStep = [500/600]; Loss: 0.0024.
Epoch: [5/5]; batchStep = [600/600]; Loss: 0.0075.
```

Training:

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

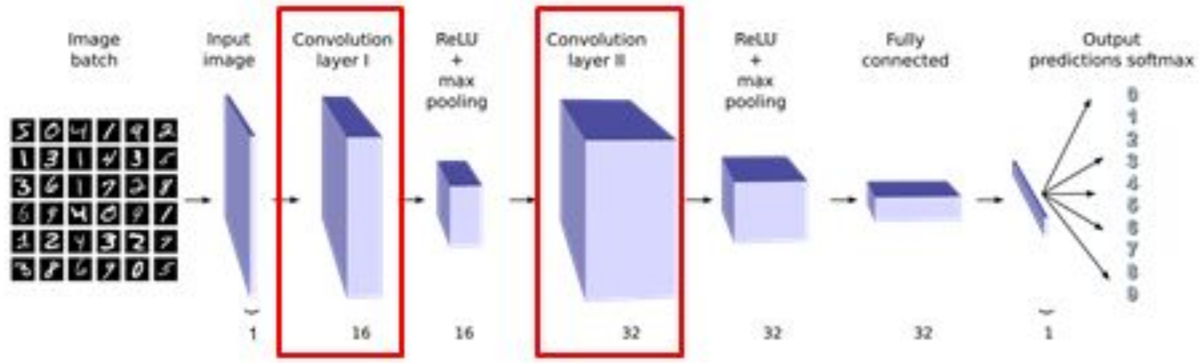
Random initial weights, mini-batches of size 100.

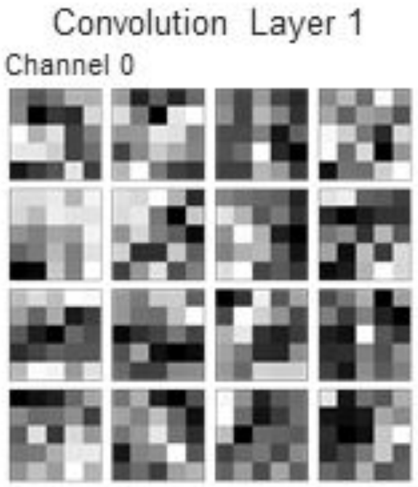
Stochastic Gradient Descent (SGD) with learning rate 10^{-3}.

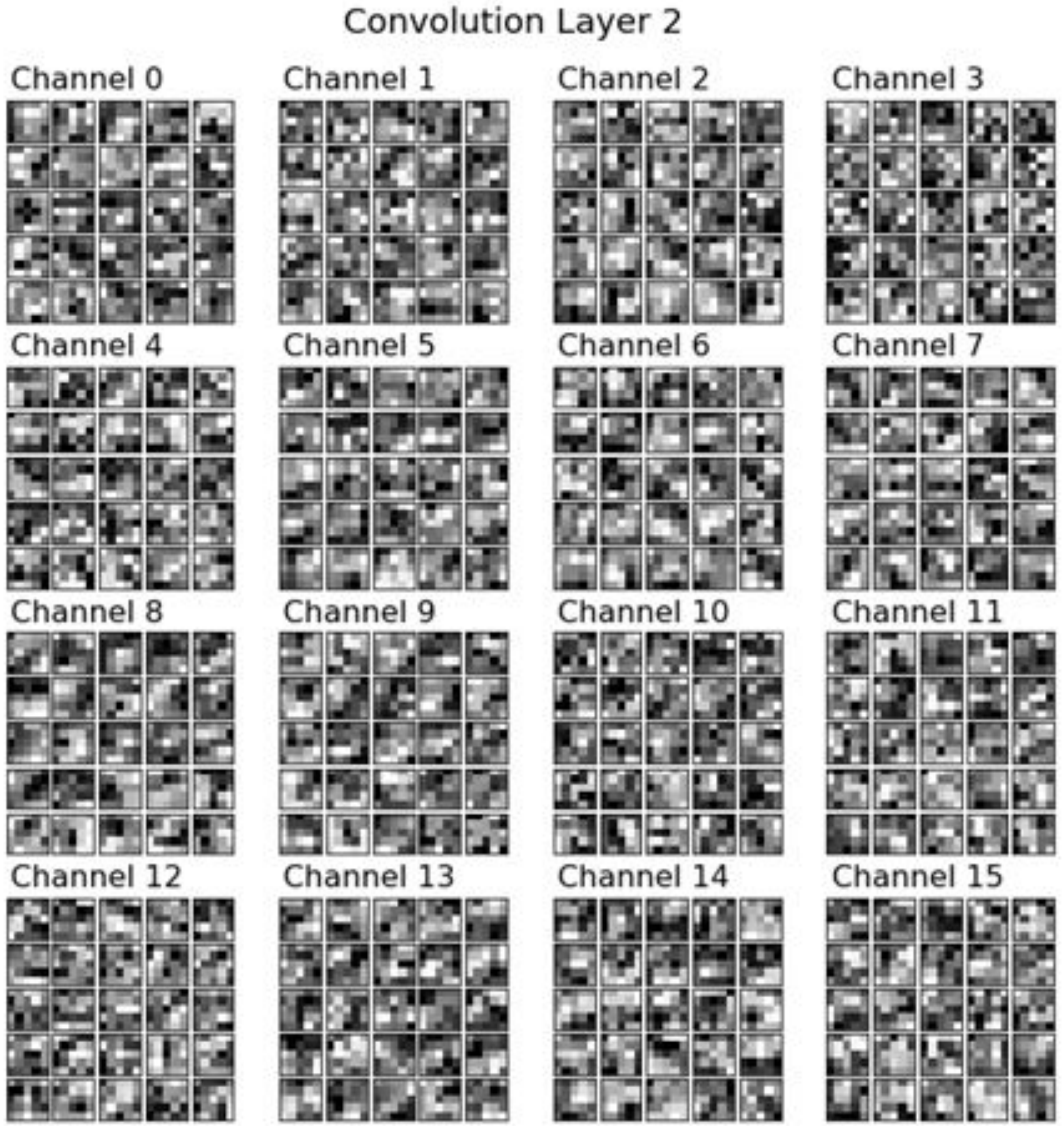
Final training loss of $7.5x10^{-3}$.

MNIST: Convolutional Neural Network

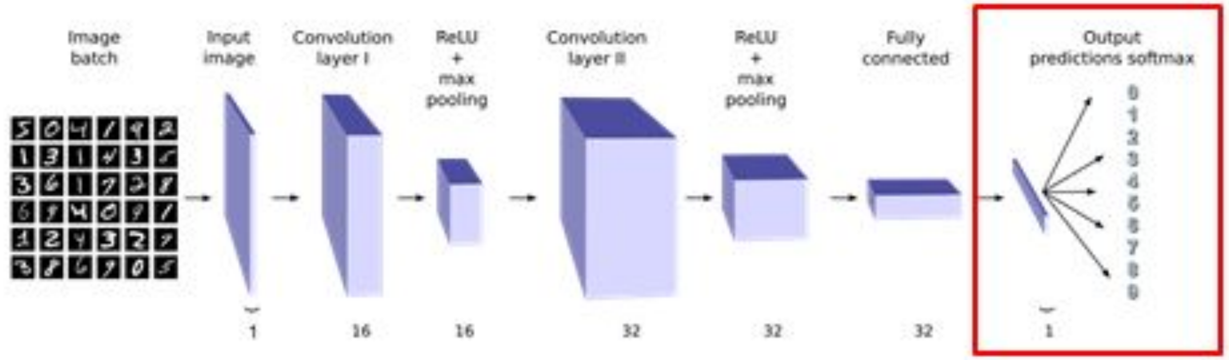
Hidden Layers







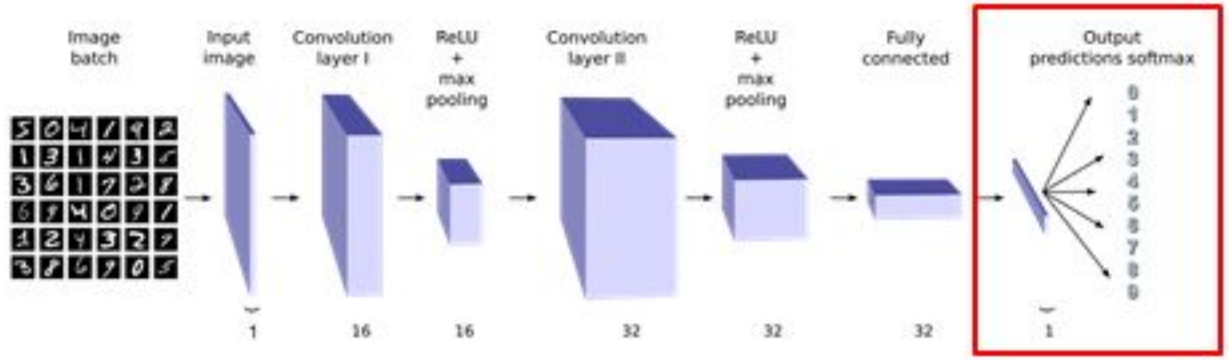
MNIST: Convolutional Neural Network



Results:

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

MNIST: Convolutional Neural Network

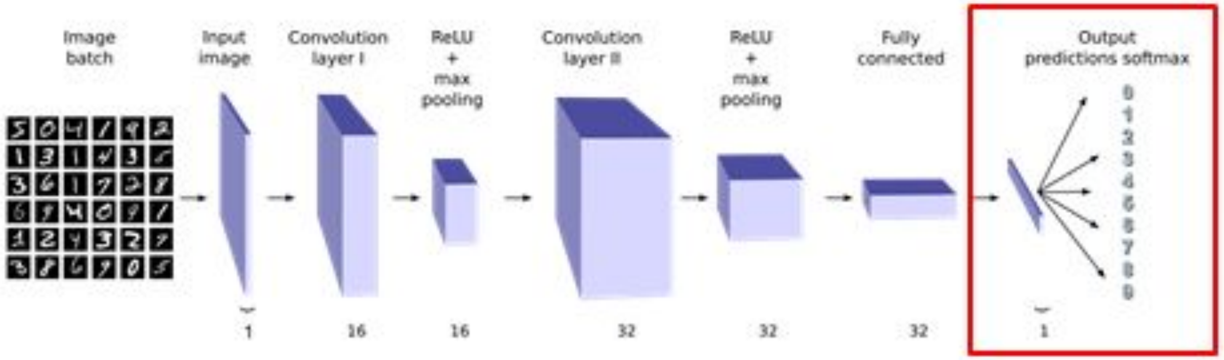


Results:

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

```
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

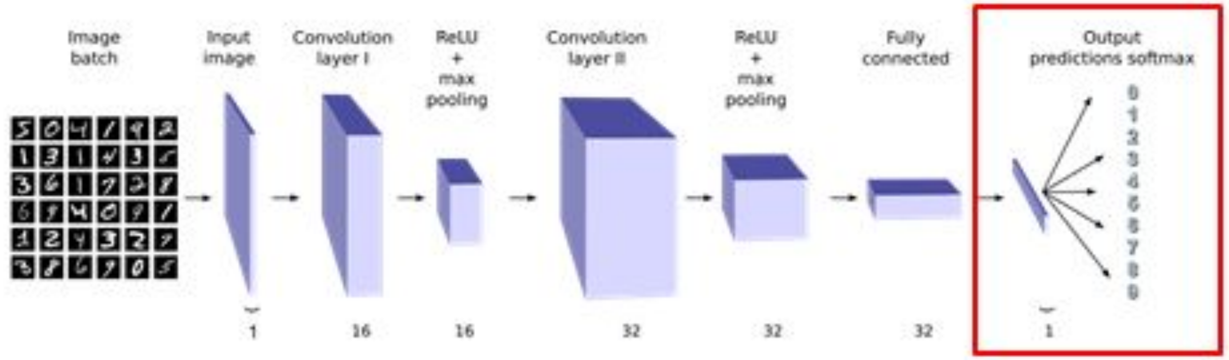


Results:

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

```
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 $L(S) = \sum_{i=1}^{m} \sum_{j=1}^{10} - y_j^{(i)} \log(\tilde{y}_j^{(i)})$

Achieves a test accuracy of 99.12%.

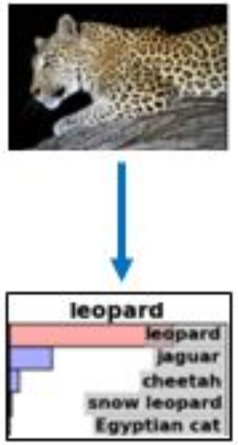
Without much tuning of the CNN, we were able to obtain very good accuracy for the MNIST dataset!

```
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%
```

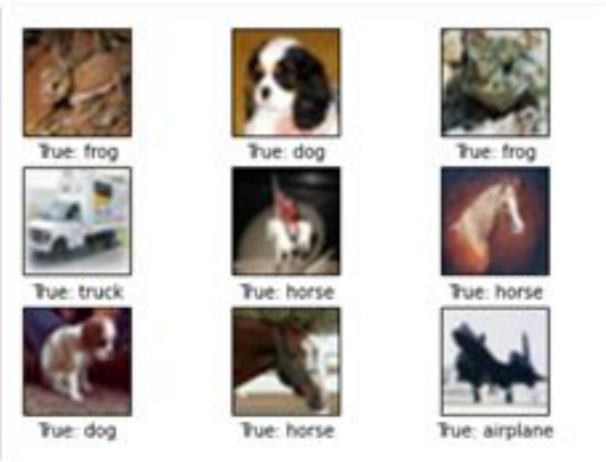
Summary

Convolutional Neural Networks (CNNs): Summary

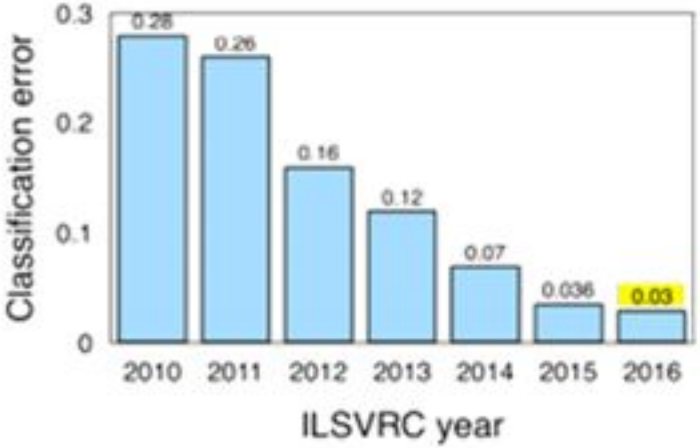
Task: Classify



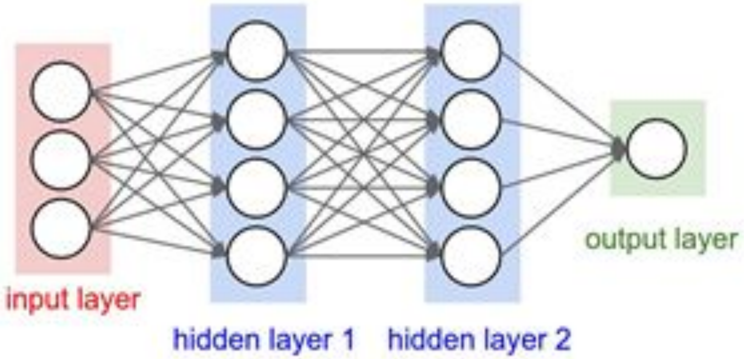
Predictions



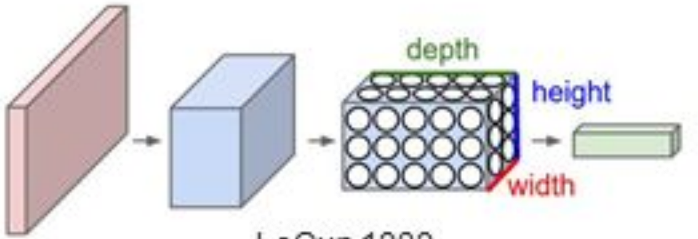
Results: ILSVRC



Neural Network (NN)

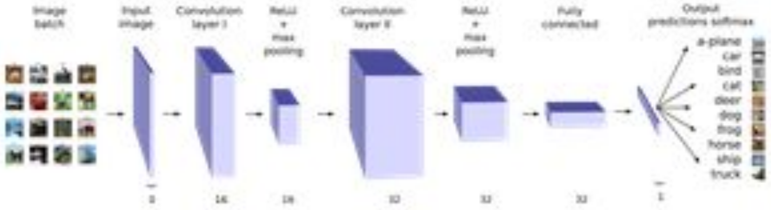


Convolutional Neural Network (CNN)



LeCun 1989

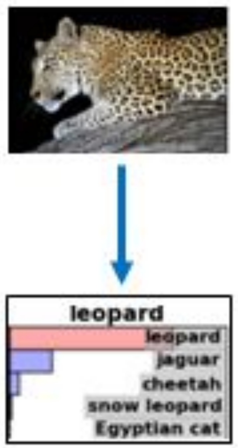
Image Classifier CNN



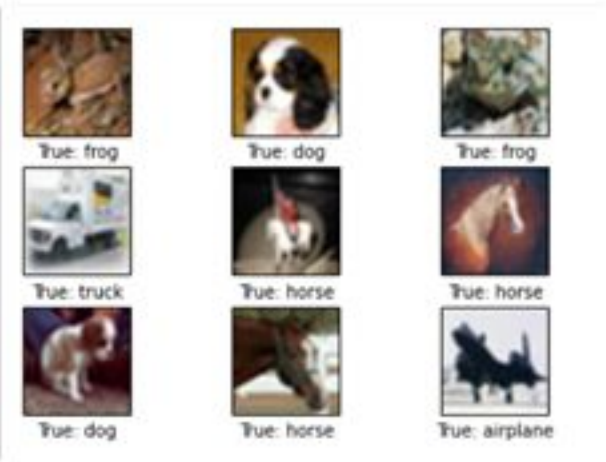
Deep neural networks are providing current state-of-the-art results.

Convolutional Neural Networks (CNNs): Summary

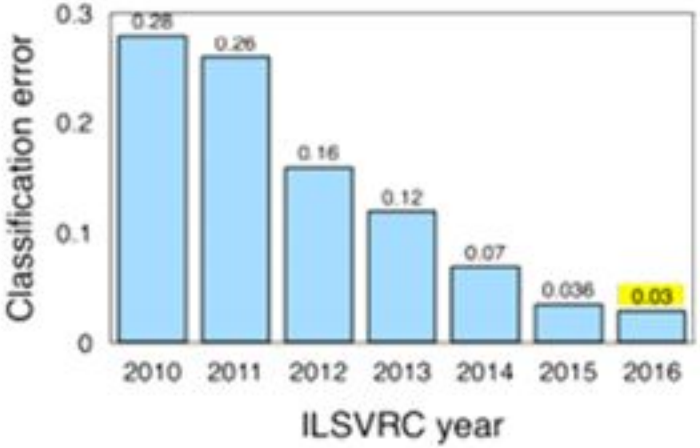
Task: Classify



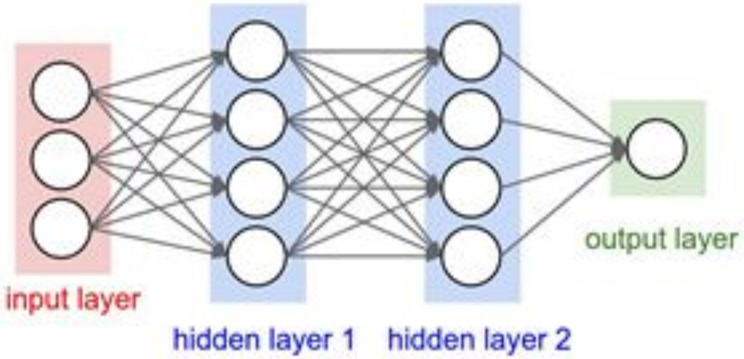
Predictions



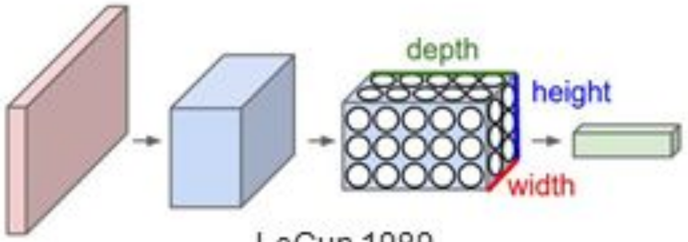
Results: ILSVRC



Neural Network (NN)



Convolutional Neural Network (CNN)

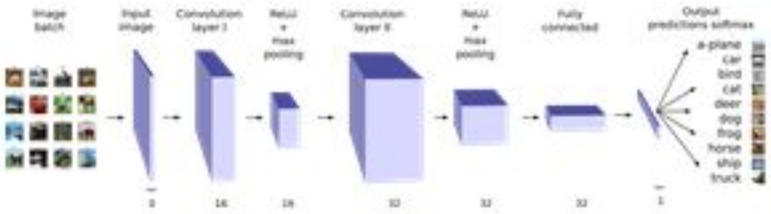


LeCun 1989

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

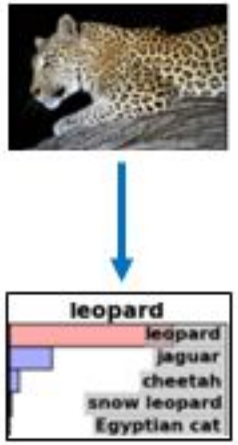
Use of shared weights provides statistical efficiency, computationally less expensive training, and possible insights into structure in data.

Image Classifier CNN



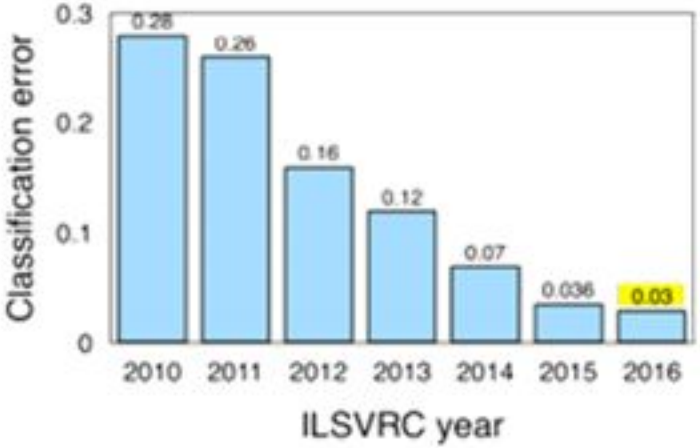
Convolutional Neural Networks (CNNs): Summary

Task: Classify

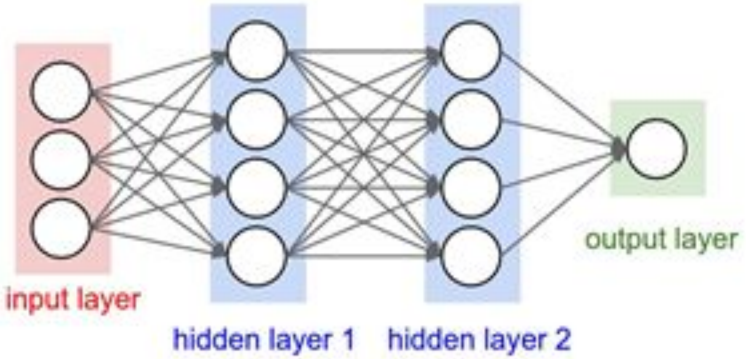


Predictions

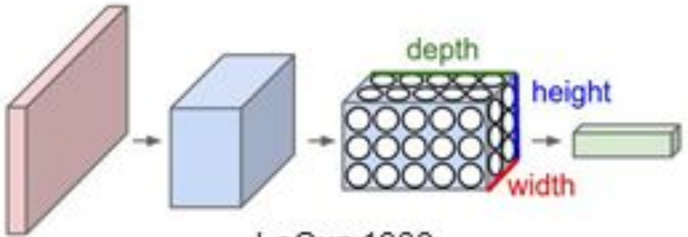
Results: ILSVRC



Neural Network (NN)

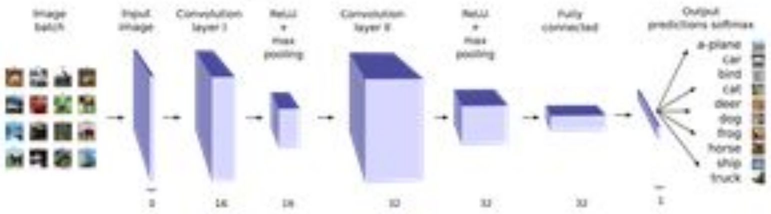


Convolutional Neural Network (CNN)



LeCun 1989

Image Classifier CNN



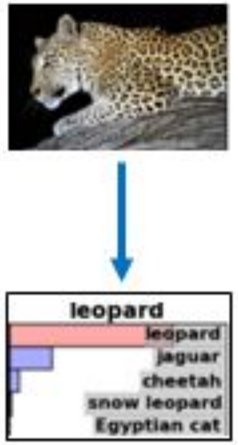
Deep neural networks are providing current state-of-the-art results.

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."

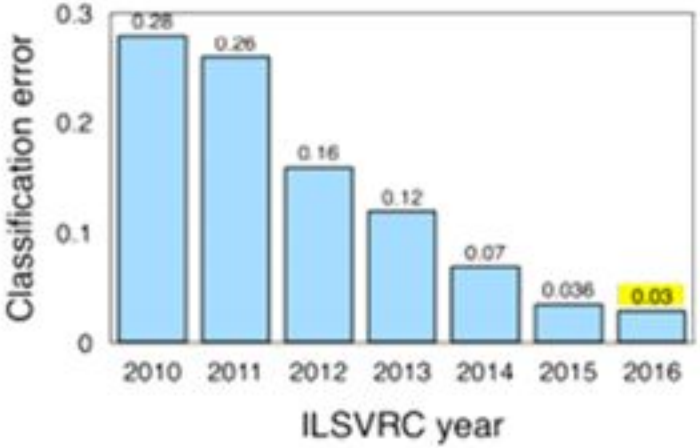
Convolutional Neural Networks (CNNs): Summary

Task: Classify

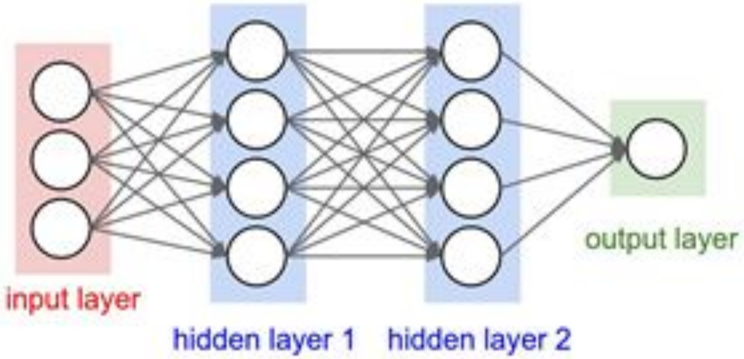


Predictions

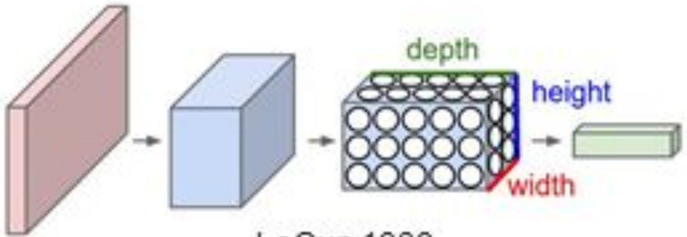
Results: ILSVRC



Neural Network (NN)

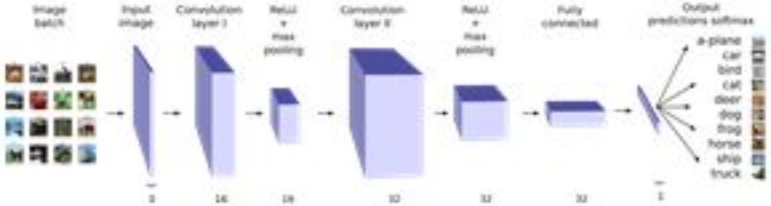


Convolutional Neural Network (CNN)



LeCun 1989

Image Classifier CNN



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."

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