EECS 553 Lecture 14
Machine Learning (ECE)
[class version]

Jeff Fessler

University of Michigan
©Laura Balzano and Jeff Fessler, UM, October 15, 2026, 00:14

F26

- Final correction:
  - **homogeneous polynomial**: $x_1^5 + 7x_1^3x_2^2 + 9x_1x_1^4$ (degree 5)
    For degree p and feature dimension d, **needs**:
    $m = \binom{p+d-1}{p} = \binom{p+d-1}{d-1}$ coefficients (including constant term).
    Previous example with $b + w_1x + w_2x^2 + w_3x^3$ was $d = 1$ and $p = 3$ so $m = \binom{d+p}{p} = \binom{4}{3} = 4 = 3 + 1$.
  - **inhomogeneous polynomial**: $c + x_1 + x_1^2 + x_1^3$ (degree 3)
    For degree-at-most p and feature dimension d, **needs**:
    $m = \binom{d+p}{p} = \binom{d+p}{d}$ coefficients.
- Why not $k(\boldsymbol{x}, \mu)$ for $\langle \boldsymbol{x}, \mu_k \rangle$?

Recap / Outline

© 2026 Fessler

- Recap
  - Kernel nearest neighbor
  - Kernel nearest centroid
  - Kernel ridge regression
  - Neural network basics
- Today
  - Neural network: Recap
    - NN loss functions
    - Backpropagation (NN training)
  - Convolutional neural networks

- Typical L-layer feedforward neural network (NN) prediction function is a **composition** of many operations:

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma_L\Big(\boldsymbol{W}^{(L)} \cdots \sigma_2\Big(\boldsymbol{W}^{(2)} \sigma_1\Big(\boldsymbol{W}^{(1)} \boldsymbol{x}\Big)\Big)\Big)$$

- We could write this as

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}} = f^{(L)} \circ f^{(L-1)} \circ \cdots f^{(2)} \circ f^{(1)},$$

  where a single layer combines a linear operation with a nonlinearity:

$$f^{(l)}(\boldsymbol{v}) = \sigma_l(\boldsymbol{W}^{(l)} \boldsymbol{v}).$$

- σ_l are nonlinear "activation" functions, typically applied element-wise
- Often $\sigma_1 = \cdots = \sigma_{L-1}$
- σ_L often **softmax** for **classification** or identity for **regression**
- $\{\boldsymbol{W}^{(l)}\}$ are matrices that linearly transform the data from one layer to the next.

Q14.1 A given feedforward NN model is (parametric, discriminative) ??

A: (T,T) B: (T,F) C: (F,T) D: (F,F) **??**

Feedforward diagram:

$$\boldsymbol{x} \xrightarrow{=\boldsymbol{z}^{(0)}} \boxed{\boldsymbol{W}^{(1)}} \xrightarrow{\boldsymbol{a}^{(1)}} \boxed{\sigma_1} \xrightarrow{\boldsymbol{z}^{(1)}} \cdots \xrightarrow{\boldsymbol{z}^{(L-1)}} \boxed{\boldsymbol{W}^{(L)}} \xrightarrow{\boldsymbol{a}^{(L)}} \boxed{\sigma_L} \xrightarrow{\boldsymbol{z}^{(L)}=} f_{\boldsymbol{\theta}}(\boldsymbol{x})$$

$\underbrace{\qquad\qquad}_{\text{layer } 1}$ $\underbrace{\qquad\qquad}_{\text{layer } L}$

NN notation: Recap

© 2026 Fessler

- Use $\boldsymbol{z}^{(\mathrm{l})}$ for output of lth layer
- Input layer:

$$\boldsymbol{z}^{(0)} = \boldsymbol{x}$$

- Hidden layer $1 \leq \mathrm{l} < L$:

$$\boldsymbol{a}^{(\mathrm{l})} = \boldsymbol{W}^{(\mathrm{l})} \boldsymbol{z}^{(\mathrm{l}-1)}$$

$$\boldsymbol{z}^{(\mathrm{l})} = \sigma_{\mathrm{l}}(\boldsymbol{a}^{(\mathrm{l})}) \qquad \text{applied element-wise.}$$

- If bias is desired, prepend a 1 to any $\boldsymbol{z}^{(\mathrm{l})}$, $0 \leq \mathrm{l} < L$, and add left column of weights to $\boldsymbol{W}^{(\mathrm{l}+1)}$
- Output layer: $\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \boldsymbol{z}^{(L)} = \boldsymbol{a}^{(L)}$ (sometimes) , namely when $\sigma_L(a) = a$, an identity activation function,
- Evaluation of the output $\hat{\boldsymbol{y}} = f(\boldsymbol{x})$ from an input $\boldsymbol{x}$ is called **forward propagation**.

- K classes so K nodes at the output layer.
- Represent classes by indicator vectors (**one-hot** encoding):

$$\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- Last layer $\boldsymbol{z}^{(L)} = \psi(\boldsymbol{a}^{(L)})$ transforms $\boldsymbol{a}^{(L)} \in \mathbb{R}^K$ to a probability vector on the **unit simplex** Δ^{K-1} in $\mathbb{R}^K$, using the **softmax function**:

$$\psi : \mathbb{R}^K \mapsto \Delta^{K-1} \subset \mathbb{R}^K, \quad \psi_k(\boldsymbol{a}) = \frac{\mathrm{e}^{a_k}}{\sum_{j=1}^K \mathrm{e}^{a_j}}, \; k = 1, \ldots, K.$$

Training the NN weights: Loss

© 2026 Fessler

- Given training data $(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)$, need to learn the network parameters aka **weights**

$$\boldsymbol{\theta} = (\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)})$$

- Typical loss functons (ERM)
  - Squared error loss (regression):

$$R(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \| f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i \|_2^2 .$$

  - **Cross-entropy** loss (for classification) with $\sigma_L =$ **softmax** ψ:

$$R(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \underbrace{y_{ik}}_{\text{one-hot}} \log[f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)]_k .$$

    Leads to faster training and better generalization than squared error for classification problems [3, p. 235].

Training the NN weights: Optimization

© 2026 Fessler

- Invariances and **data augmentation**
- We typically apply some version of **SGD** to $R(\boldsymbol{\theta})$, *e.g.*, **Adam** (adaptive moment estimation) [4]
- Selecting learning rates (step sizes) is an art...
- Improving optimization methods is an active research area, *e.g.*, Muon [5] and Scion [6].
- The "G" in SGD is *gradient* (often subgradient in practice)

Chain rule: Warm up / review

© 2026 Fessler

Before discussing backpropagation, review the chain rule.
Consider this function composition:
$f(x) = \log(3 + \sin(x^4)) = f_3(f_2(f_1(x)))$ i.e., $f = f_3 \circ f_2 \circ f_1$
$f_3(z) = \log(z),\ f_2(y) = 3 + \sin(y),\ f_1(x) = x^4$

The **chain rule** is central to **automatic differentiation** (aka autodiff or autograd) for NN training and nonlinear optimization:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f_3}{\mathrm{d}f_2} \cdot \frac{\mathrm{d}f_2}{\mathrm{d}f_1} \cdot \frac{\mathrm{d}f_1}{\mathrm{d}x} =$$

$$=$$

Backpropagation (NN training)

© 2026 Fessler

TL;DR version: **backpropagation** is an efficient means computing gradients using the chain rule [1, p. 434].

- Tweak notation i vs n $R(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N}$
  R_n: loss associated with training sample $\boldsymbol{x}_n, \; n = 1, \ldots, N$
- Focus on computing
- An efficient algorithm for computing this gradient is called **backpropagation**
- More notation:

Symbol	meaning
$w_{ij}^{(l)}$	(i,j) entry of $\boldsymbol{W}^{(l)}$
$a_{ni}^{(l)}$	ith entry of $\boldsymbol{a}^{(l)}$ when input is $\boldsymbol{x}_n$
$z_{ni}^{(l)}$	ith entry of $\boldsymbol{z}^{(l)}$ when input is $\boldsymbol{x}_n$

- From definitions:

$$\boldsymbol{a}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{z}^{(l-1)} \implies a_{ni}^{(l)} = \sum_j w_{ij}^{(l)} z_{nj}^{(l-1)}$$

$$\boldsymbol{z}^{(l)} = \sigma_l(\boldsymbol{a}^{(l)}) \implies z_{ni}^{(l)} = \sigma_l(a_{ni}^{(l)}) \text{ (element-wise)} .$$

- These quantities are known (at current value of $\boldsymbol{\theta}$) via **forward propagation** starting from $l = 1$ and working forward to $l = L$.

- $\nabla R_n(\boldsymbol{\theta})$ consists of elements

- By the **chain rule** for function composition:

$$\frac{\partial R_n(\boldsymbol{\theta})}{\partial w_{ij}^{(l)}} = \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{ni}^{(l)}} \frac{\partial a_{ni}^{(l)}}{\partial w_{ij}^{(l)}} = \delta_{ni}^{(l)} z_{nj}^{(l-1)},$$

where

$$\delta_{ni}^{(l)} \triangleq \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{ni}^{(l)}}.$$

The main idea behind the **backpropagation** algorithm is to compute $\delta_{ni}^{(L)}$ starting at the *output* layer, $l = L$ and to work *backwards* toward the input layer.

Q14.2 By the chain rule for partial derivatives,

$$\delta_{ni}^{(l)} = \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{ni}^{(l)}} = \quad ?$$

Below, k indexes the nodes in layer $l+1$.

A: $\sum_k \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{nk}^{(l)}}$

B: $\sum_k \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{nk}^{(l+1)}} \frac{\partial a_{nk}^{(l+1)}}{\partial a_{ni}^{(l)}}$

C: $\sum_k \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{nk}^{(l+1)}} \frac{\partial a_{nk}^{(l)}}{\partial a_{ni}^{(l+1)}}$

D: none of these

- Applying the definition of $\delta_{nk}^{(l)}$ to the preceding answer at $l+1$:

$$\delta_{ni}^{(l)} = \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{ni}^{(l)}} = \sum_k \delta_{nk}^{(l+1)} \frac{\partial a_{nk}^{(l+1)}}{\partial a_{ni}^{(l)}}.$$

- Because

$$a_{nk}^{(l+1)} = \sum_j w_{kj}^{(l+1)} z_{nj}^{(l)} = \sum_j w_{kj}^{(l+1)} \sigma_l(a_{nj}^{(l)}),$$

  we have (using **Lagrange's notation**):

$$\frac{\partial a_{nk}^{(l+1)}}{\partial a_{ni}^{(l)}} =$$

- Combining gives the recursion for the δ_{ni} terms:

$$\delta_{ni}^{(l)} = \sum_k \delta_{nk}^{(l+1)} w_{ki}^{(l+1)} \sigma_l'(a_{ni}^{(l)}).$$

- It remains to determine $\delta_{ni}^{(L)}$.
- Here it depends on the loss function
- Consider the simplest regression case
  - squared error loss
  - scalar outputs y_n
  - identity output activation $\sigma^{(L)}(a) = a$

$$R_n(\boldsymbol{\theta}) = (f_{\boldsymbol{\theta}}(\boldsymbol{x}_n) - y_n)^2 = \left(z_{n1}^{(L)} - y_n\right)^2 = \left(a_{n1}^{(L)} - y_n\right)^2,$$

so

$$\delta_{n1}^{(L)} = \frac{\partial R_n(\boldsymbol{\theta})}{\partial a_{n1}^{(L)}} = 2\left(a_{n1}^{(L)} - y_n\right).$$

- Classification:
  - Cross-entropy loss
  - σ_l is softmax
-

$$R_n(\boldsymbol{\theta}) = -\sum_k y_{nk} \log[f_{\boldsymbol{\theta}}(\boldsymbol{x}_n)]_k = -\sum_k y_{nk} \log z_{nk}^{(L)}$$
$$= -\sum_k y_{nk} \log \psi_k(\boldsymbol{a}_n^{(L)}),$$

so

$$\delta_{ni}^{(L)} = -\sum_k y_{nk} \frac{1}{\psi_k(\boldsymbol{a}_n^{(L)})} \frac{\partial \psi_k(\boldsymbol{a}_n^{(L)})}{\partial a_{ni}^{(L)}}.$$

Now use this property of the derivative of softmax:

$$\frac{\partial \psi_k(\boldsymbol{a})}{\partial a_i} = \begin{cases} \psi_k(\boldsymbol{a})(1 - \psi_i(\boldsymbol{a})), & i = k \\ -\psi_k(\boldsymbol{a})\psi_i(\boldsymbol{a}), & i \neq k. \end{cases}$$

Backpropagation - summary

© 2026 Fessler

Forward pass:

Using current weights $\boldsymbol{\theta}$, compute $f(\boldsymbol{x}_n)$ and *store* intermediate values $a_{ni}^{(l)}, z_{nj}^{(l)}$

Initialize backward pass:

For $i = 1$ to d_L

Compute $\delta_{ni}^{(L)}$ (using loss function)

End

Backward pass:

For $l = L - 1$ down to 1

For $i = 1$ to d_l

$\delta_{ni}^{(l)} = \sum_k \delta_{nk}^{(l+1)} w_{ki}^{(l+1)} \sigma_l'(a_{ni}^{(l)})$

For $j = 1$ to d_{l-1}

$\frac{\partial R_n(\boldsymbol{\theta})}{\partial w_{ij}^{(l)}} \leftarrow \delta_{ni}^{(l)} z_{nj}^{(l-1)}$

End

End

End

Optimization options

- Use this gradient of R_n in SGD:
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha_t \nabla R_n(\boldsymbol{\theta}_{t-1})$
- Or repeat for every n and accumulate to get full gradient $\nabla R(\boldsymbol{\theta}_{t-1})$.
- Or combine multiple samples in a **mini-batch** to exploit GPU SIMD.

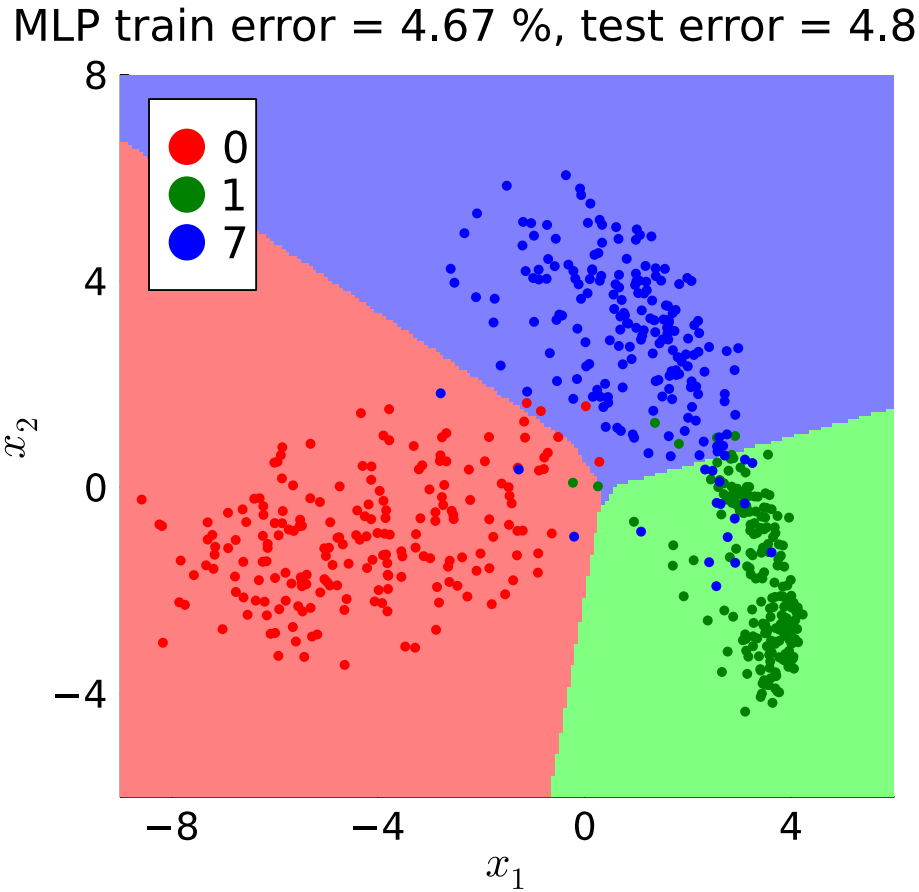
Deep feedforward network remarks

- Note: each hidden unit passes and receives information to and from only those units to which it is connected $\implies$ parallel implementation.
- In general, $R(\boldsymbol{\theta})$ will have several local minima. Because the overall cost function is nonconvex, convergence to a global minimizer is not guaranteed.
- To avoid overfitting, regularization is often used.
- Initializing $\boldsymbol{\theta}_0$: typically small iid Gaussian, often with layer-dependent variances (Kaiming He, 2015) [7], or sometimes random orthogonal matrices.
- When training a NN with backpropagation of the squared error loss, the resulting network is a LS fit to the Bayes discriminant function [2, p. 303].
- **universal approximation theorems** state that sufficiently large NNs (like basic MLPs) can approximate any continuous function
- Everything discussed so far has been understood since the 1980s

Demo with "plain JULIA code" - no ML frameworks

Model: 2-layer MLP: $2 \to 10 \to \text{ReLU} \to 3 \to \text{softmax}$

- $K = 3$ MNIST digits
- $d = 2$ PCA features
- cross-entropy loss (cf multinomial logistic regression)
- Tikhinov regularization
- Quasi-Newton optimizer with `ForwardDiff`



- Many image processing deep networks use
  We call these
- Examples of CNNs: LeNet, AlexNet, VGG, Inception, **ResNet**, **U-Net**, ...
- Especially useful for datatypes where convolutions would be a natural signal processing operation, like digital images, digital audio, digital video, ...
- A conventional MLP model would not easily exploit the spatial/temporal relationships of such signals.
- Typical input is a **data tensor** (*i.e.*, an array) of size $N_x \times N_y \times 3$.
  So, in our notation: $d = 3N_xN_y$.
- For classification problems, output is a softmax probability vector in the simplex Δ^{K-1}.

ImageNet classification

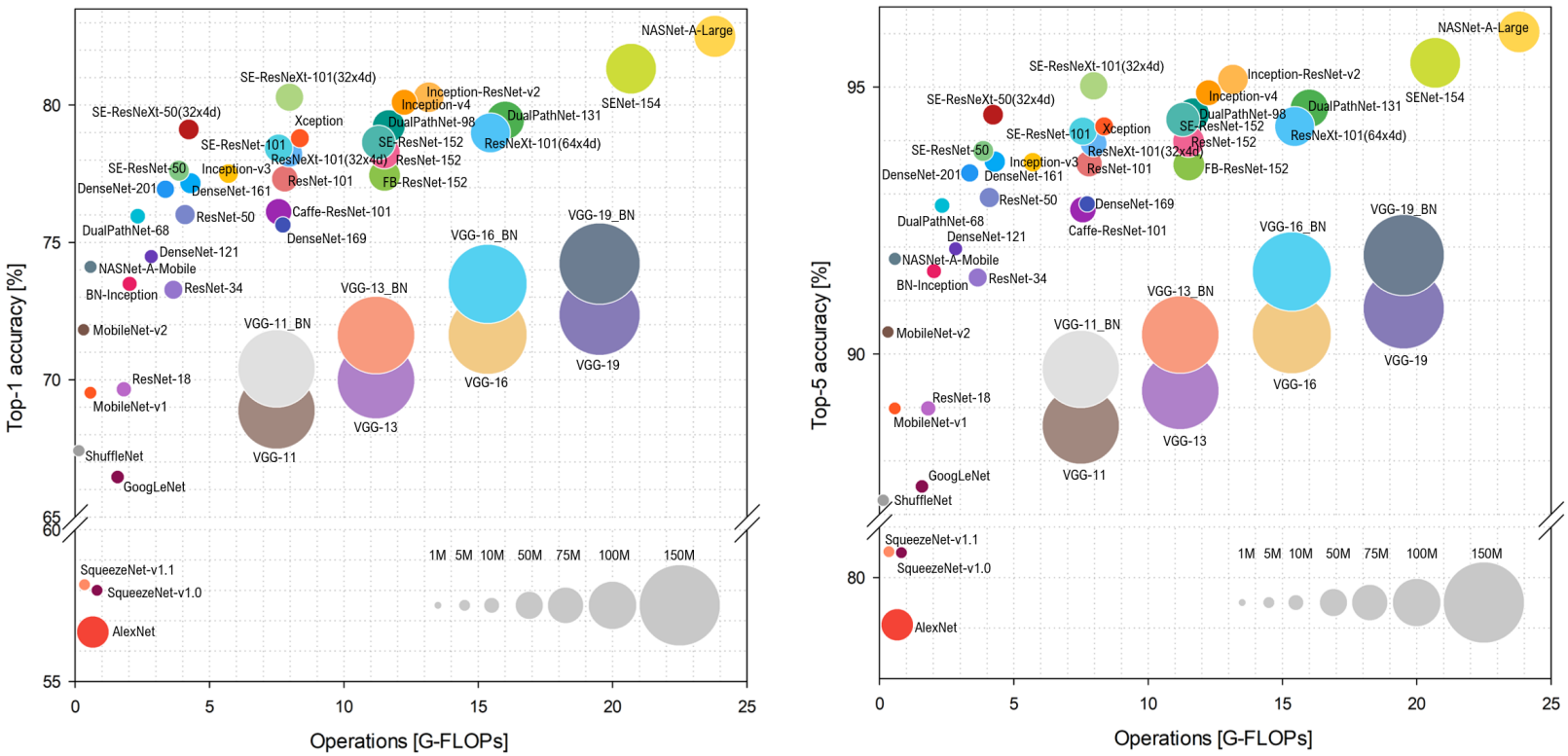
© 2026 Fessler

ImageNet is a database of 14M labeled images of $> 20,000$ categories envisioned by Fei-Fei Li in 2006.

- Annual image recognition challenge uses a "trimmed" list of $K = 1000$ non-overlapping classes.
- Hierarchical categories (mammal, dog, poodle, ...)
- Color RGB images with varying sizes.
- Typically standardized to some fixed resolution for training / testing.
- Perhaps 6-10% of the labels are incorrect!
- Human error rate with training is about 5% for Top 5 accuracy.

CNN classification accuracy

© 2026 Fessler



(a) (b)

FIGURE 1: Ball chart reporting the Top-1 and Top-5 accuracy *vs.* computational complexity. Top-1 and Top-5 accuracy using only the center crop versus floating-point operations (FLOPs) required for a single forward pass are reported. The size of each ball corresponds to the model complexity. (a) Top-1; (b) Top-5.

Figure shows accuracy on the "ImageNet" dataset; courtesy [8].

Top-1: **0-1 loss**; Top-5: true label is among the highest 5 softmax probabilities.

Convolutional layers

© 2026 Fessler

- A convolutional layer has a weight matrix $\boldsymbol{W}$ that has a particular sparse and **banded** structure,
- When the input image is vectorized, then the weight matrix has structure, so that each row of $\boldsymbol{W}$ only operates on
- The exact structure of $\boldsymbol{W}$ depends on the chosen boundary conditions or padding mode.

We can most easily visualize what this structure does by considering each row of $\boldsymbol{W}$ as a that operates on a

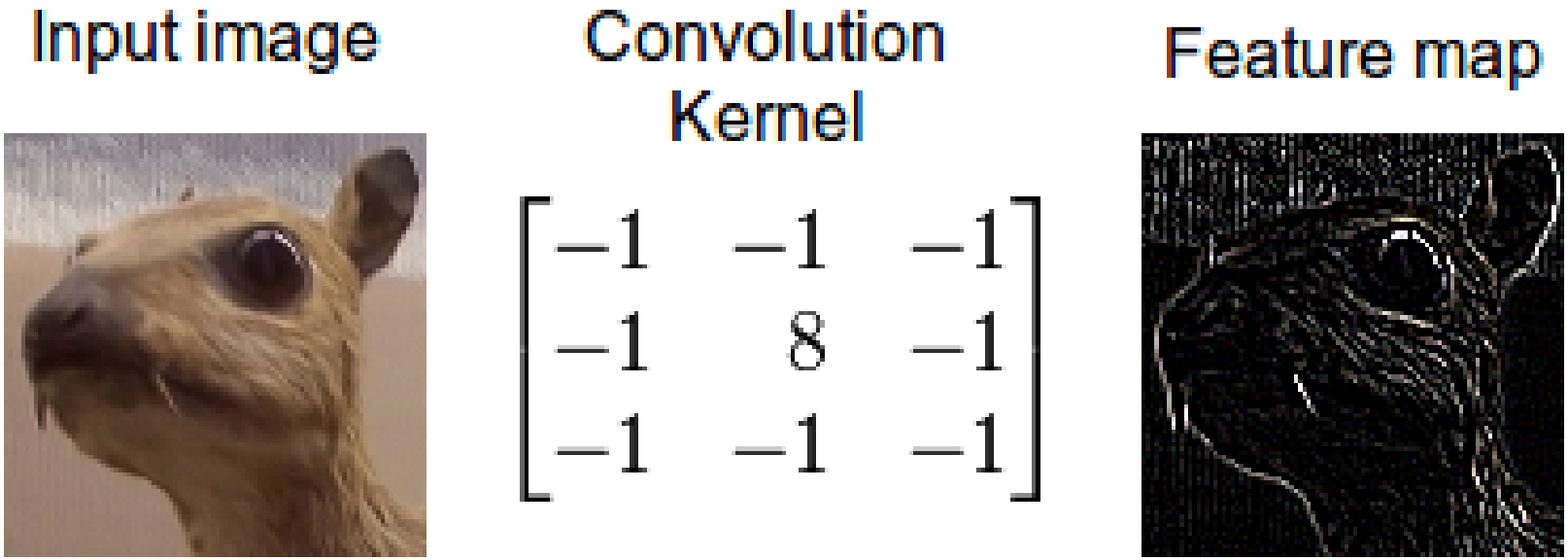


Figure courtesy [9].
Note "kernel" here has a different meaning, *i.e.*, the convolution mask. There are several types of kernels in ML [1, §16.3.1]! See this blog.
This kernel is just for illustration; in ML the kernel values are learned.
See the **kernel** page for more examples.

The convolution mask gets applied as a sliding window (and in each row of $\boldsymbol{W}$, the same mask is positioned to apply to different pixels of the vectorized image).

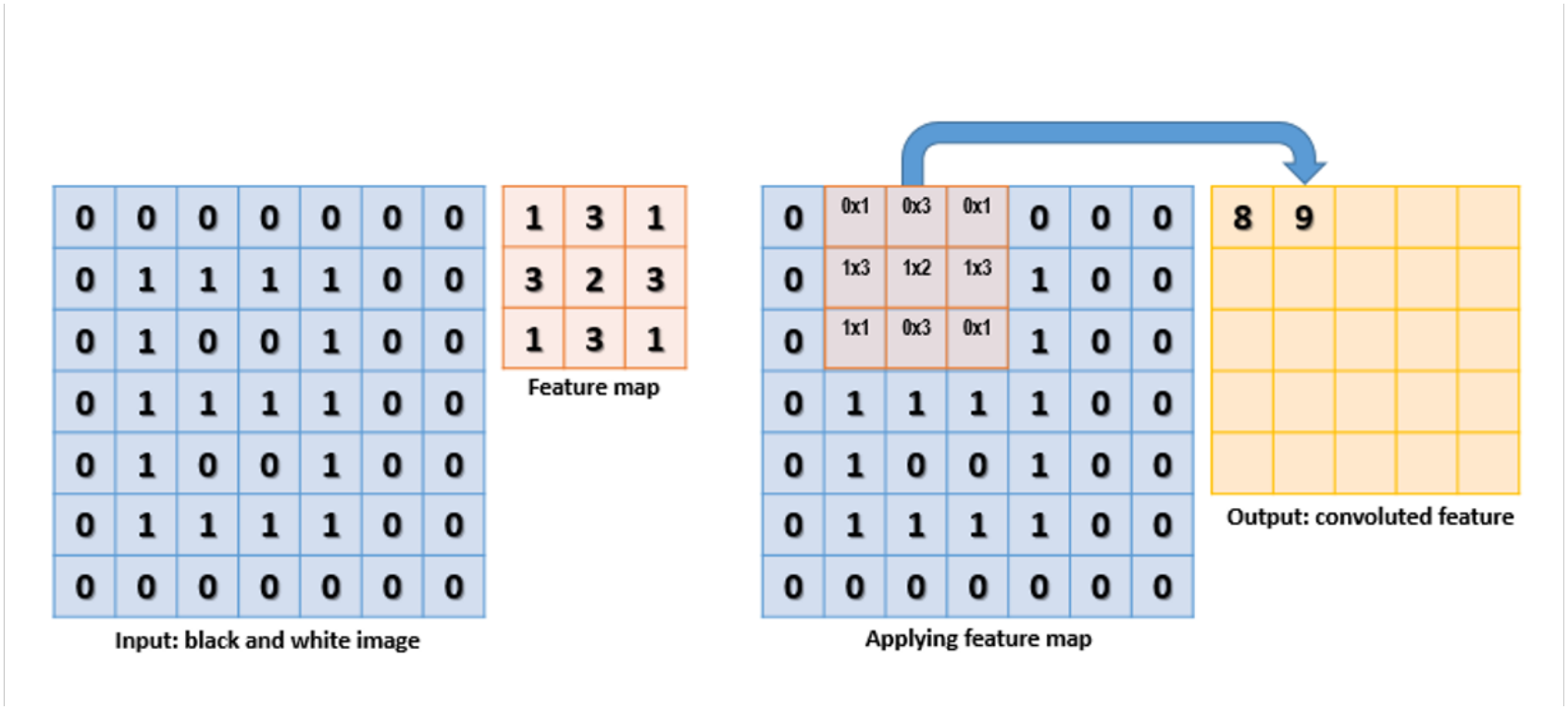
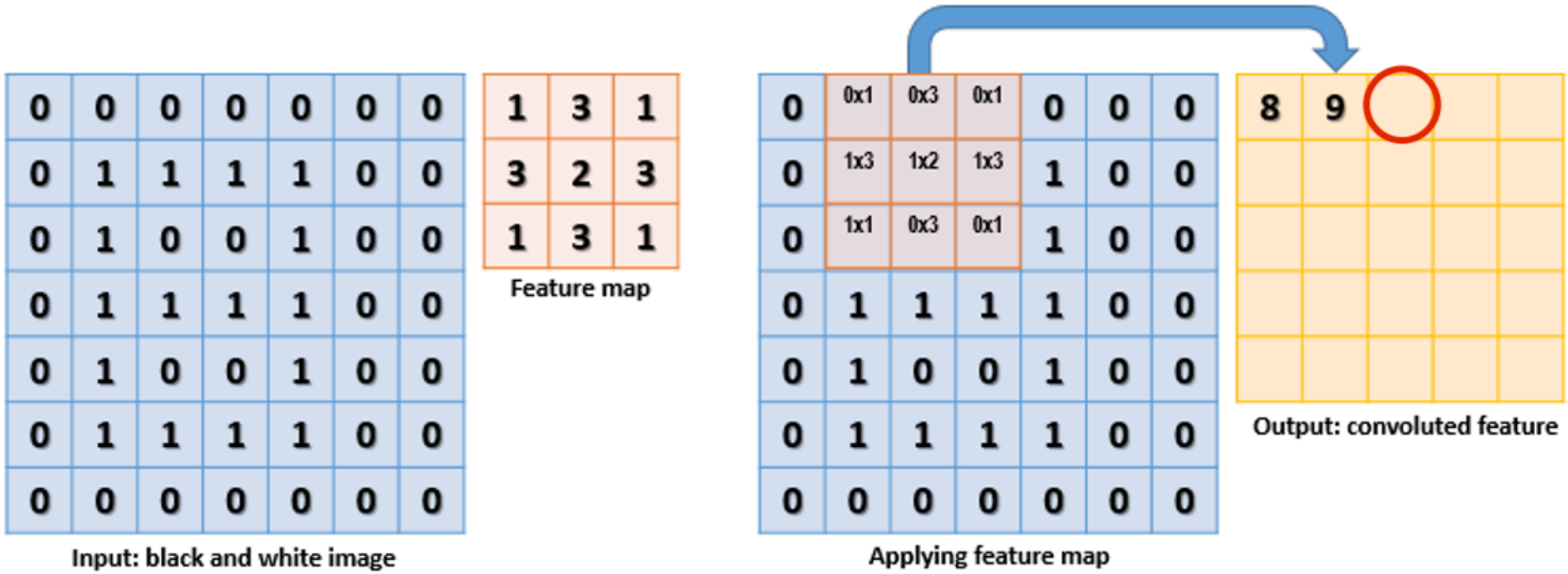


Figure courtesy Clay Scott.

CQ: Convolution

© 2026 Fessler



Q14.3 What is the next entry (in the red circle)?

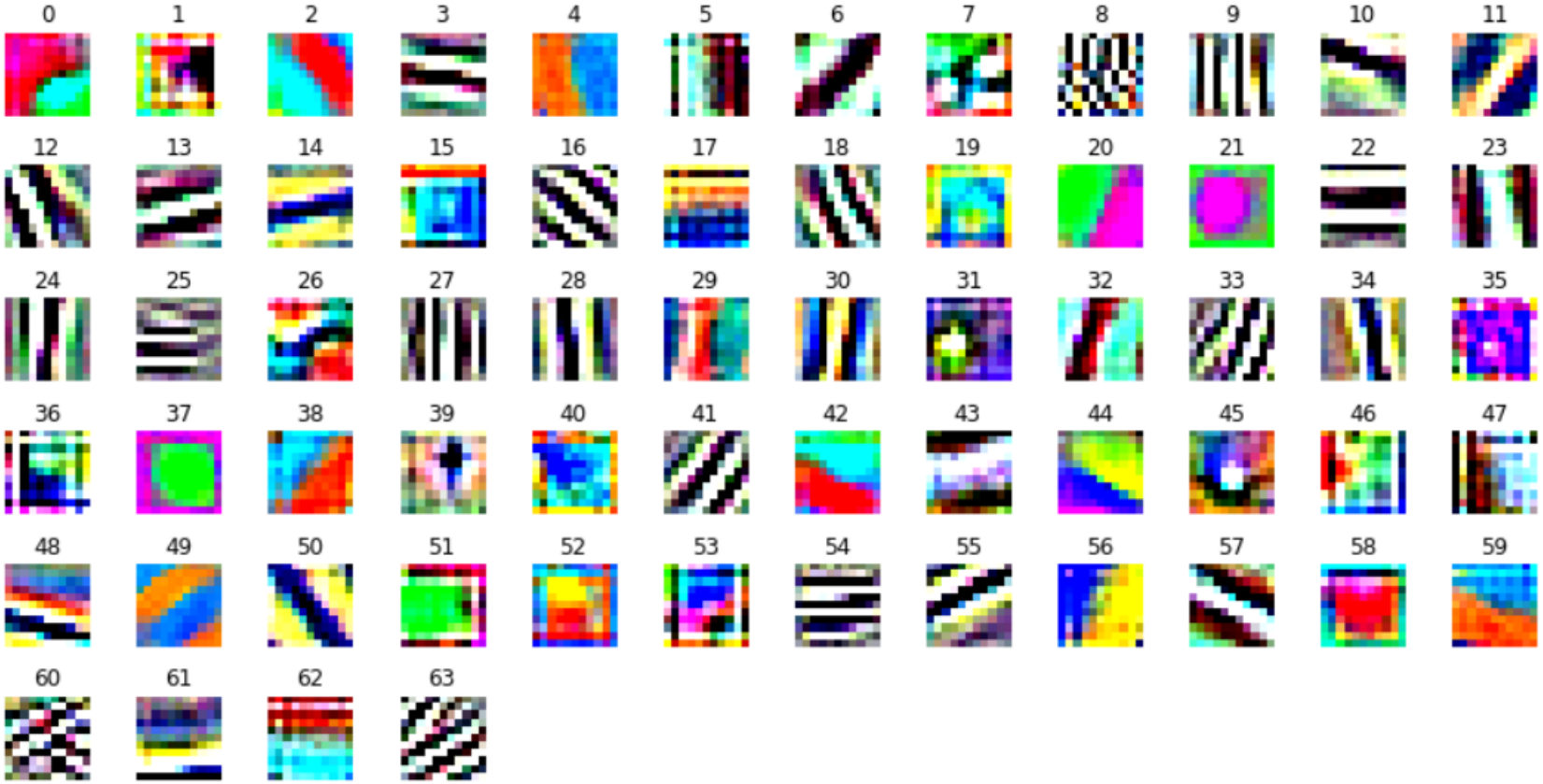
A: 4 B: 8 C: 9 D: 10 E: none of these

Convolutional layers

© 2026 Fessler

- Several filters per layer
- Filter coefficients are the weights to be learned
- # of weights in layer =
- Useful for data with temporal or spatial structure, *e.g.*, images
- Not fully connected + shared weights $\implies$ far fewer weights to learn than a MLP

We can visualize the filters learned by a network, *e.g.*, AlexNet:



First layer features of AlexNet, figure courtesy [10].

Pooling/downsampling layer

© 2026 Fessler

- CNNs often combine convolutional layers with **pooling layers**.
- Goal is to
- No weights to learn in pooling layer
- Common implementation for images:
  2×2 window, stride of 2 (no parameters to learn)

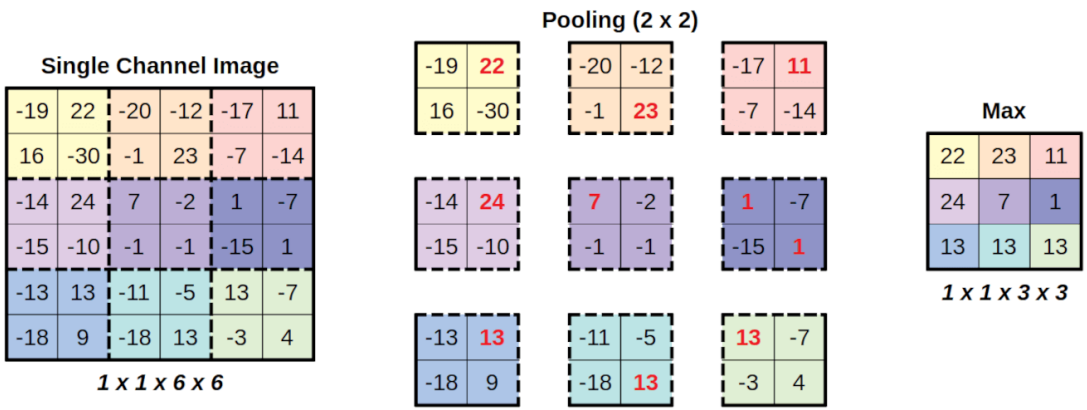


Image courtesy Daniel Voigt Godoy

- Combine convolutional, pooling, and fully connected layers
- Backprop extends naturally to CNNs
- Major breakthrough in 1998: LeNet5 [11] for handwritten digit recognition
- CNNs are used in face recognition systems, self driving cars, many other object recognition systems
- Speech recognition uses conformer that combines CNN (local) with **transformer** (global) models

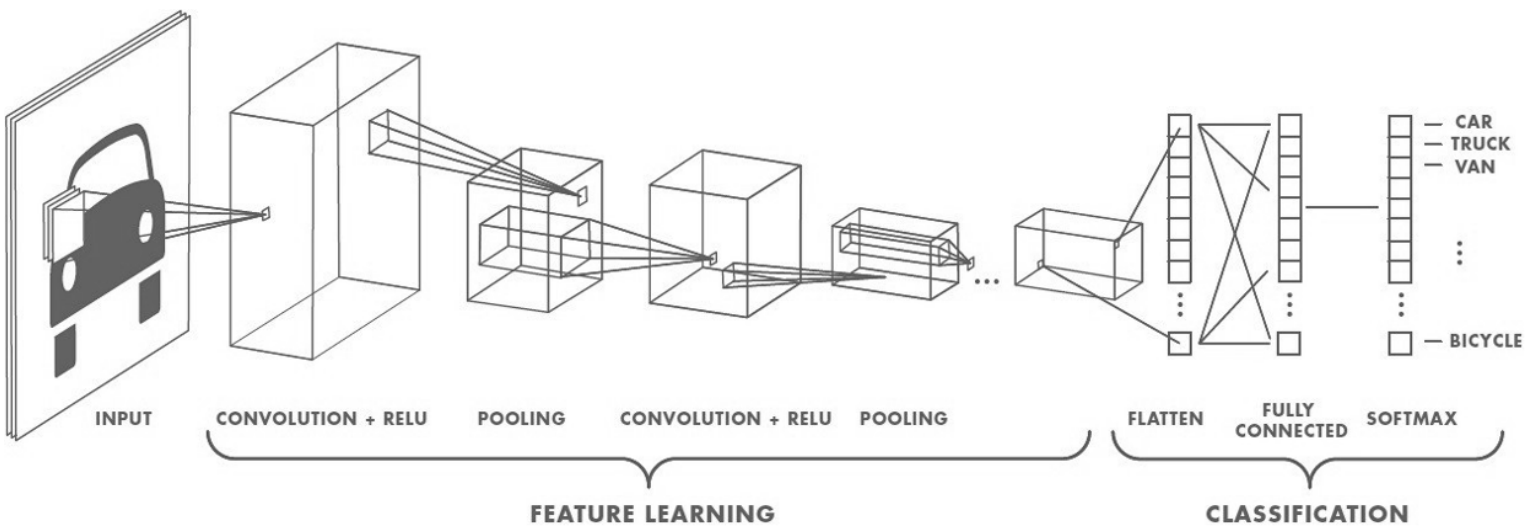
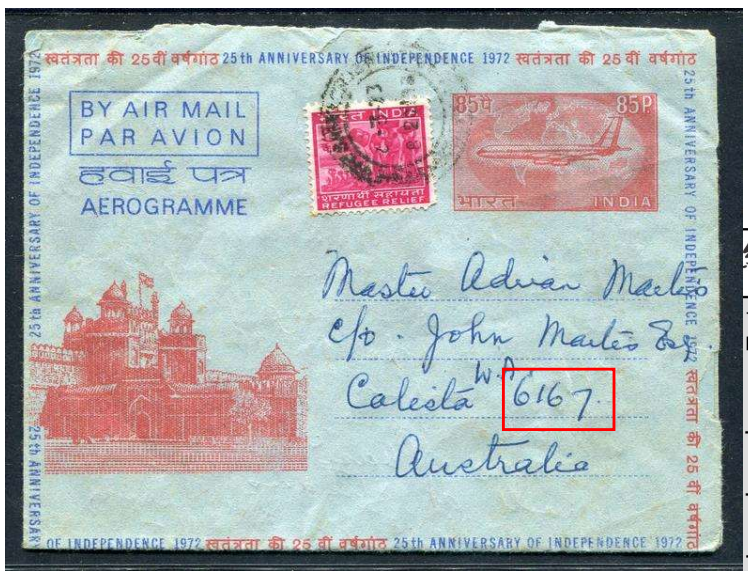


Figure courtesy [12].

Some CNN history

© 2026 Fessler

- Handwriting recognition was an important problem for banks and the post office.
- AT&T Labs had a project focused on this task.





Images courtesy Bernhard Kainz, ICL.

MNIST is a dataset of handwritten numbers created in 1994 to help researchers develop and benchmark/compare methods.

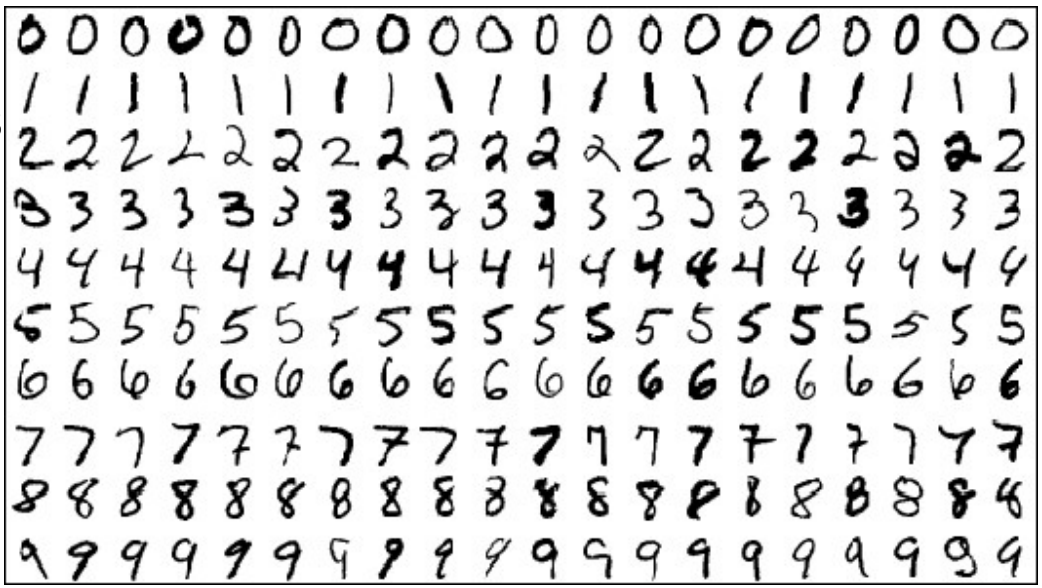
Cantered and scaled

50.000 training samples

10.000 test samples

28 x 28 images

10 classes



Images courtesy Bernhard Kainz, ICL.

- Demo by Yann LeCun in 1989 of digit recognition using a CNN: youtube url.  The video was restored using a CNN!

U-Net architecture

© 2026 Fessler

The **U-Net** architecture combines CNN, pooling / up-sampling, and specific **skip connections**

- global context and local details
- helps gradients propagate, improving training

Won the 2015 ISBI image segmentation challenge

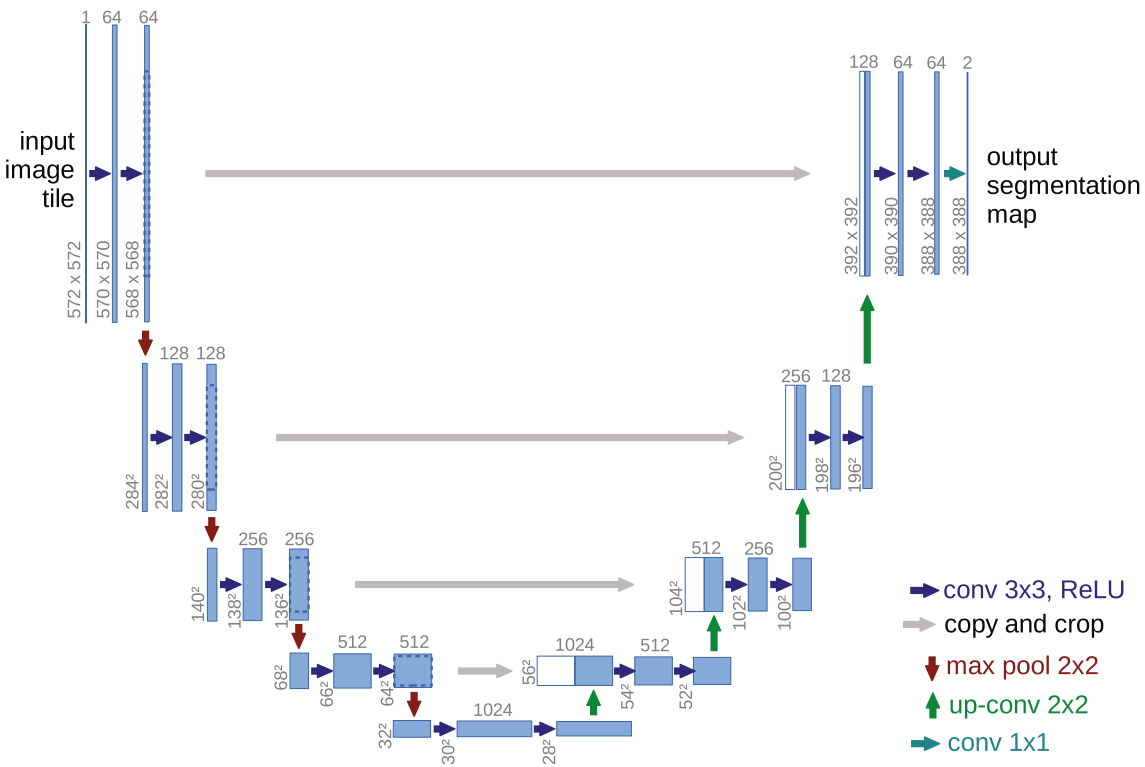


Figure from [13]

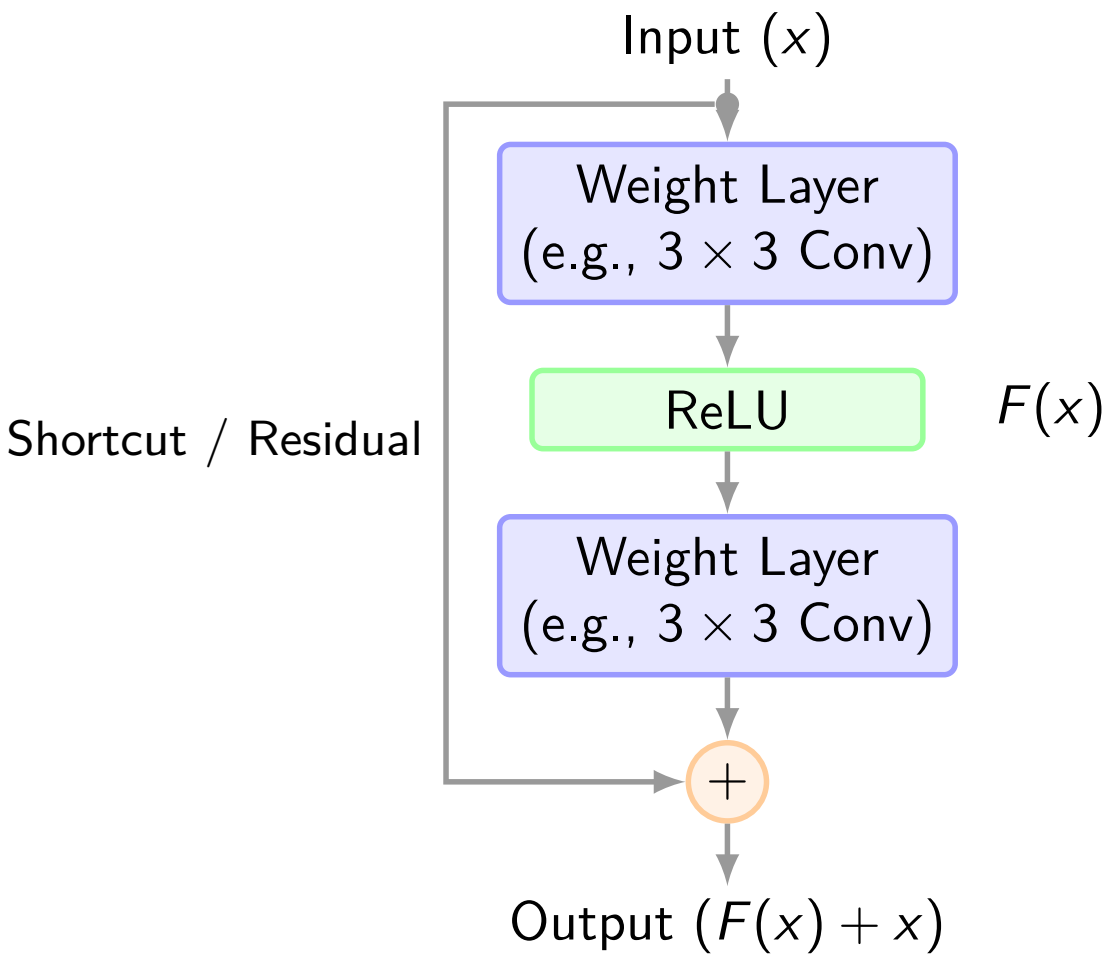
Other important NN architectures

© 2026 Fessler

- Recurrent Neural Networks (RNN)
- Long Short Term Memory (LSTM)
- Autoencoders
- Generative Adversarial Networks (GAN)
- Residual NN **ResNet**
- Transformers (Attention + MLP)
- Mixture of Experts (Attention + many smaller MLPs) [1, §13.6.2]
- Many more ...

ResNet block

© 2026 Fessler



Tixz help from Gemini
Example: Image denoising

Some history

© 2026 Fessler

- CNNs have been around since the 1990s, autoencoders and RNNs since the 1980s,
- For many years, neural networks were not considered state of the art for many prediction tasks
- Larger neural networks needed for greater accuracy, but they can be difficult to train
  - Vanishing gradients problem (large gradients at output layer are reduced to small gradients at early layers)
  - More (bad) local minima, backprop gets stuck
  - Not enough data to prevent overfitting
- Then neural networks experienced a renaissance that began around 2006

Enabled by ...

- Graphics processing units (GPUs): parallel floating-point calculators with 100s-1000s of cores
- Large, public databases such as ImageNet [14] that has over 14 million *labeled* examples and 20 thousand classes of objects
- Refined training strategies
  - Dropout
  - Modifications of SGD (*e.g.*, Adam)
  - Batch normalization
- Rectified linear unit ReLU (for vanishing gradient problem)
  Many alternatives: GELU, SwiGLU, SiLU/Swish, Leaky ReLU...
- Improved design (how to define the layers)
- Real-world examples?

References I

[1] K. P. Murphy.
Probabilistic machine learning: An introduction.
MIT Press, 2022.

[2] R. O. Duda, P. E. Hart, and D. G. Stork.
Pattern classification.
Wiley, New York, 2001.

[3] C. Bishop.
Pattern recognition and machine learning.
Springer, 2006.

[4] D. P. Kingma and J. Ba.
Adam: A method for stochastic optimization, 2014.

[5] K. Jordan.
Muon: An optimizer for hidden layers in neural networks, 2024.

[6] T. Pethick, W. Xie, K. Antonakopoulos, Z. Zhu, A. Silveti-Falls, and V. Cevher.
Training deep learning models with norm-constrained LMOs.
In *Proc. Intl. Conf. Mach. Learn*, 2025.

[7] K. He, X. Zhang, S. Ren, and J. Sun.
Delving deep into rectifiers: surpassing human-level performance on imageNet classification.
In *Proc. Intl. Conf. Comp. Vision*, pages 1026–34, 2015.

[8] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano.
Benchmark analysis of representative deep neural network architectures.
IEEE access, 6:64270–64277, 2018.

[9] Tim Dettmers.
Deep learning in a nutshell: Core concepts, 2015.
`https://developer.nvidia.com/discover/convolution`.

References II

[10] Niranjan Kumar.
Visualizing convolution neural networks using pytorch, 2019.
https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324, 1998.

[12] Jinde Shubham.
What exactly does cnn see?, 2018.
https://becominghuman.ai/what-exactly-does-cnn-see-4d436d8e6e52.

[13] O. Ronneberger, P. Fischer, and T. Brox.
U-net: convolutional networks for biomedical image segmentation.
In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–41, 2015.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei.
Imagenet: A large-scale hierarchical image database.
In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.