Convolutional Neural Networks

Tianxiang (Adam) Gao

Feb 6, 2025

Outline

Recap: Neural Networks and Training

- MLPs are **parameterized** function $f_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta} = \{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\}$:
  - Forward Propagation (biases omitted): Start with $\boldsymbol{x}^0 = \boldsymbol{x}$

$$\begin{aligned} \boldsymbol{z}^\ell &= \boldsymbol{W}^\ell \boldsymbol{x}^{\ell-1}, \quad \forall \ell \in \{0, 1, 2, \ldots, L\} \\ \boldsymbol{x}^\ell &= \phi(\boldsymbol{z}^\ell), \end{aligned}$$

  - Backward Propagation (biases omitted): Start with $d\boldsymbol{z}^L = (\boldsymbol{x}^L - \boldsymbol{y}) \odot \phi'(\boldsymbol{z}^L)$

$$\begin{aligned} d\boldsymbol{z}^\ell &= \left[(\boldsymbol{W}^{\ell+1})^\top d\boldsymbol{z}^{\ell+1} \right] \odot \phi'(\boldsymbol{z}^\ell), \quad \forall \ell \in \{1, 2, \ldots, L-1\} \\ d\boldsymbol{W}^\ell &= d\boldsymbol{z}^\ell \boldsymbol{x}^{(\ell-1)\top} \end{aligned}$$

- The training involves solving an **optimization** problem to iteratively update the $\boldsymbol{\theta}$

$$\min_{\boldsymbol{\theta}} \quad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) := R_S(f_{\boldsymbol{\theta}}),$$

  where ℓ is a **loss** function and $\mathcal{S} := \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{\ell}$ is a **training set**.
- This optimization problem can be solved using **gradient**-based methods such as *(stochastic) gradient descent (SGD)*, *gradient descent with momentum*, *RMSProp*, *Adam*, etc:

$$\boldsymbol{\theta}^+ = \boldsymbol{\theta} - \eta \cdot \boldsymbol{v}^+,$$

  where $\eta > 0$ is a **learning rate** and $\boldsymbol{v}$ is a **search direction**.

Recap: Generalization and Regularization

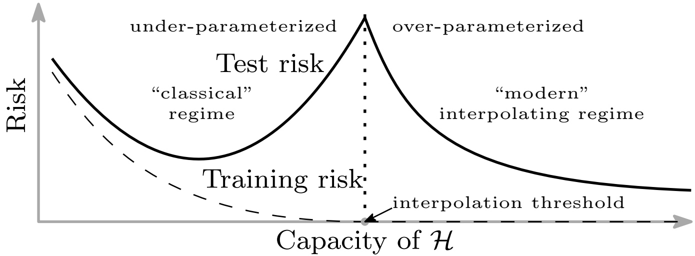
- **Model complexity trade-off**: The expected risk $R(f_S) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\ell(f_S(\boldsymbol{x}), \boldsymbol{y})$ is upper bounded:

$$R(f_S) \leq R_S(f_S) + \mathfrak{R}_S(\mathcal{H}) + \tilde{\mathcal{O}}(n^{-1}).$$

- **Bias-Variance trade-off**: The expectation of $R(f_S)$ over random sample S is decomposed as:

$$\mathbb{E}_S[R(f_S)] = \underbrace{\mathbb{E}_S(f_S - \bar{f})^2}_{\textbf{Variance term}} + \underbrace{\mathbb{E}_{\mathcal{D}}(\bar{f} - f^*)^2}_{\textbf{Bias term}} + \underbrace{R(f^*)}_{\textbf{irreducible}}$$

  where $\bar{f} := \mathbb{E}_S[f_S]$ and f^* is the optimal hypothesis.
- **Regularization**: Weight decay, dropout regularization, and stochastic weight averaging
- **Hyperparameter tune**: Validation set, random search, log scale
- **Overparameterization**: Double descent, flat minimum, implicit regularization

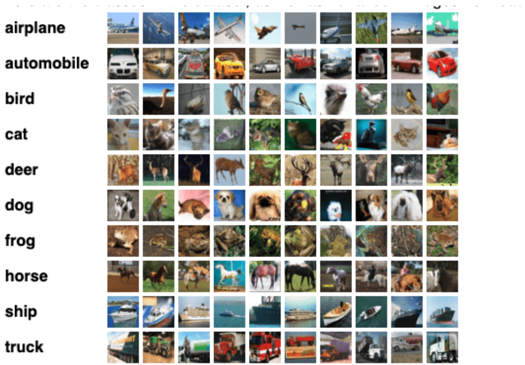


Outline

Image Classification

- **Input**: An image
- **Output**: Cat? Binary classification (0 or 1).

Multiple Classification: Softmax

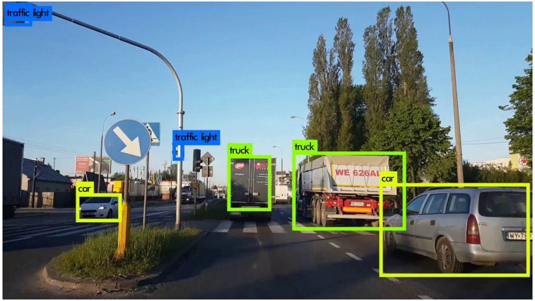


- **Input**: An image
- **Output**: Class label $\{0, 1, 2, \cdots, 9\}$.
- **Softmax**: Converts a vector $\boldsymbol{z}$ of **logits** into probability distribution across classes

$$\text{Softmax}(\boldsymbol{z}_i) = \frac{e^{\boldsymbol{z}_i}}{\sum_{j=1}^{C} e^{\boldsymbol{z}_j}},$$

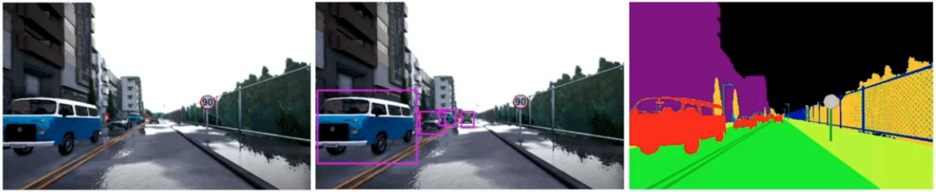
where C is the number of classes.

Object Detection



- **Input**: An image
- **Outputs**:
  - Class label
  - **Bounding box**: $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$
  - Confidence scores: A probability or confidence score between 0 and 1.

Semantic Segmentation



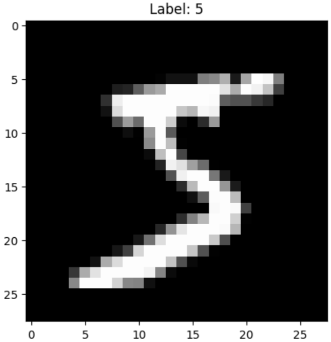
Input image Object Detection Semantic Segmentation

- **Input**: An image
- **Outputs**:
  - A pixel-wise classification map
  - Each pixel is assigned a **class label**
  - The output is the same spatial size as the input image

Neural Style Transfer

- **Input**: Content image, style image
- **Output**: Generated image

Challenges in Image Data: High Dimensionality

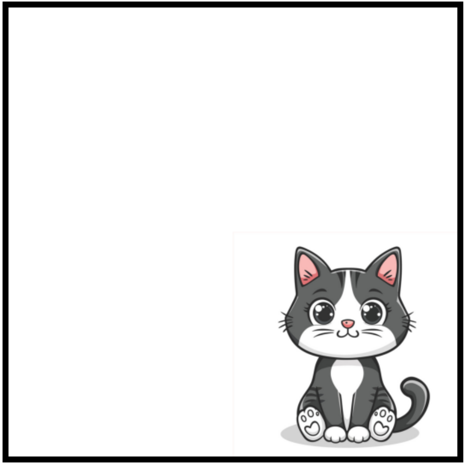


$28 \times 28 \times 1 = 784$

$1099 \times 733 \times 3 \approx 2.5$ million pixels

- A **two-layer** neural network with width 1000 leads to **3 billion parameters** to train.
- Despite having large datasets, the limited *computational cost* makes training challenging.

Translation Invariance in Images



Key Insight

Image features (edges, textures, or objects) can appear **anywhere** in the image, but they retain the same meaning regardless of their position. This is known as **translation invariance**.

Importance of Spatial Structure in Images

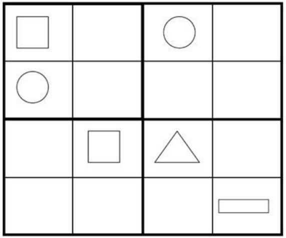
Key Insight

The **spatial structure** and local connectivity of pixels define an image's recognizable features. When the spatial arrangement is disrupted, the image loses its recognizable form.

Outline

Filters and Edge Detection in Image Processing

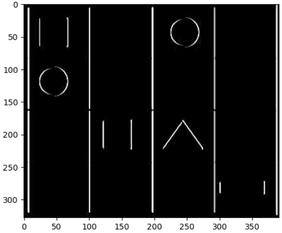
Filter: Filters are small matrices that are used to **detect** certain patterns, such as edges, textures, or other important features from the input data.

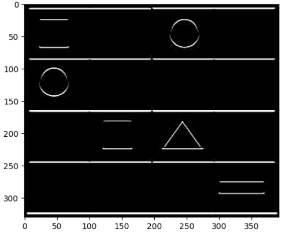


Original Image

$$\underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{Vertical Filter}}$$

$$\underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Horizontal Filter}}$$





Convolution Operation

Define: In image processing, the **convolution operation** slides a small **filter** over the input image, performing a **locally linear transformation** (*i.e.*, element-wise multiplication and summing the results) to produce a feature map that detects patterns.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\textbf{input image } 6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\textbf{filter } 3 \times 3} = \underbrace{\begin{bmatrix} \quad & \quad & \quad & \quad \\ & & & \\ & & & \\ & & & \end{bmatrix}}_{\textbf{feature map } 4 \times 4}$$

Convolution Operation

Define: In image processing, the convolution operation slides a small **filter** over the input image, performing a **locally linear transformation** (*i.e.*, element-wise multiplication and summing the results) to produce a feature map that detects patterns.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\textbf{input image } 6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\textbf{filter } 3 \times 3} = \underbrace{\begin{bmatrix} -5 & -4 & 0 & 8 \\ 10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}}_{\textbf{feature map } 4 \times 4}$$

- If the input image is $n \times n$ and the filter size is $f \times f$, then the output feature map has size $(n - f + 1) \times (n - f + 1)$.

Padding

Define: Padding refers to adding extra pixels (usually *zeros*) around the input data to control the size of the output feature map.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 1 & 2 & 7 & 4 & 0 \\ 0 & 1 & 5 & 8 & 9 & 3 & 1 & 0 \\ 0 & 2 & 7 & 2 & 5 & 1 & 3 & 0 \\ 0 & 0 & 1 & 3 & 1 & 7 & 8 & 0 \\ 0 & 4 & 2 & 1 & 6 & 2 & 8 & 0 \\ 0 & 2 & 4 & 5 & 2 & 3 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Preserving Spatial Dimensions**: Padding maintains the spatial dimensions in deeper neural networks.
- **Capture Edge information**: Padding prevents the loss of boundary information during convolution.
- **Controlling Output Size**: Padding helps ensure feature maps retain the required size for subsequent layers.
- The shape of feature map: $(n + 2p - f + 1) \times (n + 2p - f + 1)$.

"Valid" and "Same" Convolution

Define: Padding refers to adding extra pixels (usually zeros) around the input data to control the size of the output feature map.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 1 & 2 & 7 & 4 & 0 \\ 0 & 1 & 5 & 8 & 9 & 3 & 1 & 0 \\ 0 & 2 & 7 & 2 & 5 & 1 & 3 & 0 \\ 0 & 0 & 1 & 3 & 1 & 7 & 8 & 0 \\ 0 & 4 & 2 & 1 & 6 & 2 & 8 & 0 \\ 0 & 2 & 4 & 5 & 2 & 3 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- **Valid**: No padding, output size is $(n - f + 1) \times (n - f + 1)$.
- **Same**: Padding ensures the output has the same shape as the input, with output size $(n + 2p - f + 1) \times (n + 2p - f + 1)$.

$$n + 2p - f + 1 = n \implies p = \frac{f-1}{2}$$

Hence, generally, filters have **odd** dimensions, *e.g.*, 3×3 or 5×5.

Stride

Define: Stride in CNNs refers to the number of pixels by which the filter moves across the input during convolution, affecting the output size by skipping certain positions.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\textbf{input image } 6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\textbf{filter } 3 \times 3} = \underbrace{\begin{bmatrix} -5 & 0 \\ 0 & -4 \end{bmatrix}}_{\textbf{feature map } 2 \times 2 \textbf{ with stride } 2}$$

- **Control Output Size:** Larger stride results in a smaller feature map.
- **Computational Efficiency:** Larger strides require fewer convolution operations.
- **Output Feature Map Shape:**

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor,$$

where $\lfloor x \rfloor$ is the floor function, returning the largest integer less than or equal to x.

Simplified Convolutional Layer

- Let $\boldsymbol{X} \in \mathbb{R}^{n \times n}$ be the input image, and $\boldsymbol{F} \in \mathbb{R}^{f \times f}$ be the **trainable** filter.
- The convolutional layer is defined as:

$$\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{F} + b, \quad \boldsymbol{A} = \text{ReLU}(\boldsymbol{Z})$$

where:
  - $b \in \mathbb{R}$ is the **bias** term added to each element in $\boldsymbol{Z}$.
  - $\boldsymbol{Z} \in \mathbb{R}^{(n-f+1)\times(n-f+1)}$ represents the **pre-activation values**, assuming no padding and a stride of 1.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\boldsymbol{X}} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\boldsymbol{F}} = \underbrace{\begin{bmatrix} -5 & -4 & 0 & 8 \\ 10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}}_{\boldsymbol{Z}}$$

Key Observation

While MLPs use explicit weight matrices, CNNs use **filters** that serve the role of weight matrices, learning specific features directly from the data.

Neurons in CNNs

- We can represent the input image $\boldsymbol{X}$ and filter $\boldsymbol{F}$ as vector forms, $\boldsymbol{x} \in \mathbb{R}^{n^2 \times 1}$ and $\boldsymbol{w} \in \mathbb{R}^{f^2 \times 1}$, by stacking their entries:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n \end{bmatrix} \implies \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{n^2} \end{bmatrix}, \quad \text{and} \quad \boldsymbol{F} = \begin{bmatrix} \boldsymbol{f}_1 & \cdots & \boldsymbol{f}_f \end{bmatrix} \implies \boldsymbol{w} = \begin{bmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{f^2} \end{bmatrix}.$$

- Thus, each convolution can be viewed as extracting a local receptive field using a *projection matrix* $\Pi_i \in \mathbb{R}^{f^2 \times n^2}$ to obtain $\hat{\boldsymbol{x}}_i$, followed by an inner product with $\boldsymbol{w}$:

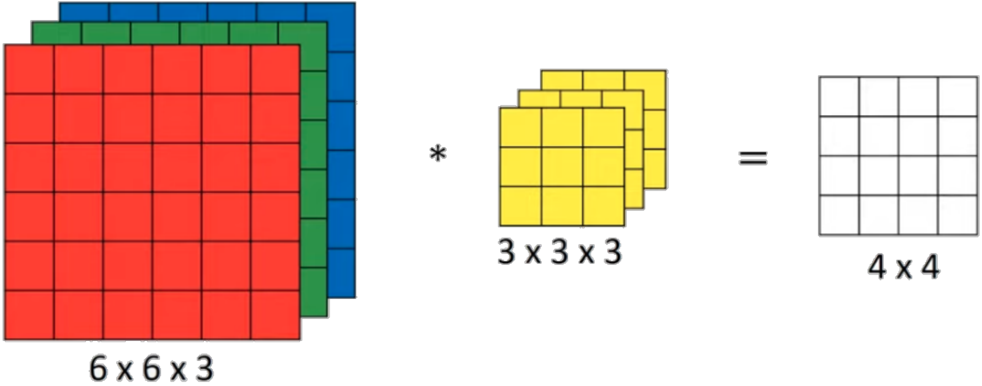
$$\hat{\boldsymbol{x}}_i = \Pi_i \boldsymbol{x}, \qquad \boldsymbol{z}_i = \boldsymbol{w}^\top \hat{\boldsymbol{x}}_i + b, \qquad \boldsymbol{a}_i = \text{ReLU}(\boldsymbol{z}_i), \quad \forall i \in \{1, 2, \ldots, (n - f + 1)^2\}$$

Key Insights

- **Sharing:** Each neuron in a convolutional layer **shares** the same weights and bias across spatial locations, reducing the impact of high dimensionality.
- **Sparsity:** Each output depends only on a **locally** small portion of the input.

Convolution Over Volumes

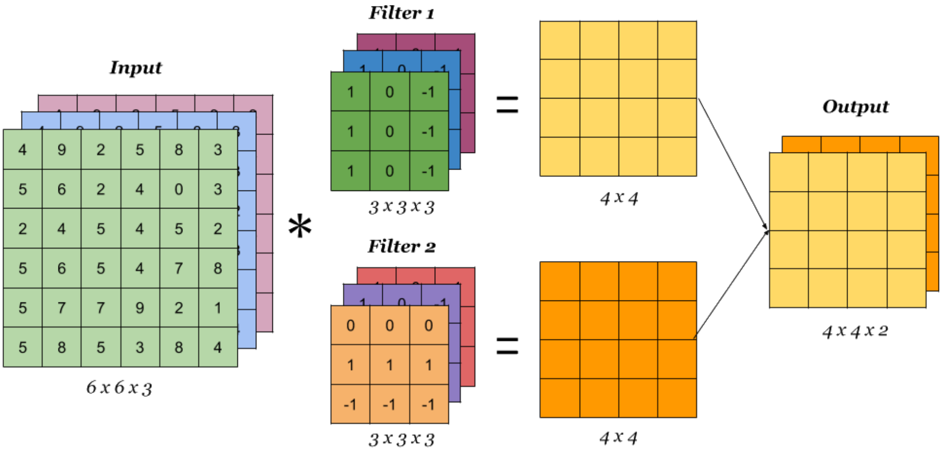
- The input can have **multiple** channels (e.g., an RGB image), and the filter must have the **same** number of channels to properly apply the convolution operation, which performs a *locally linear transformation*.
- The filter has size $n_H \times n_W \times n_C$



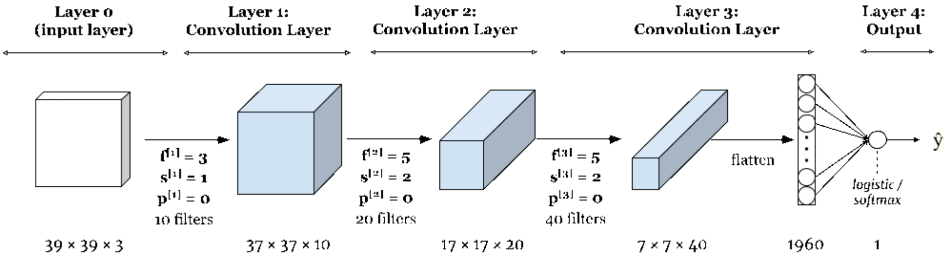
Andrew Ng, "Convolutional Neural Networks". Coursera.

Convolution with Multiple Filters

- **Multiple** filters can be used in a convolution layer to detect multiple features.



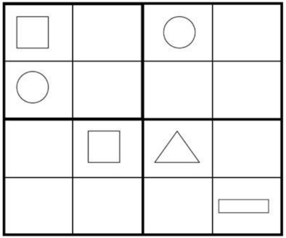
Simple CNN Example



- The output of CNN is **flattened** into a vector
- The flattened vector serves as the input to a **fully connected layer**
- In CNN design, feature maps typically shrink in spatial size while channels increase as depth grows.

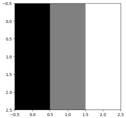
Feature Map as an Indicator

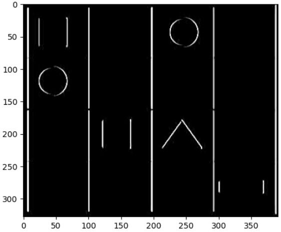
Feature Map as an Indicator: The output feature map highlights detected patterns, with higher values indicating matched regions.



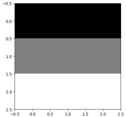
Original Image

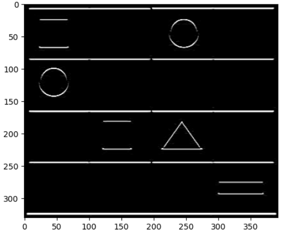
$$\underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{Vertical Filter}}$$





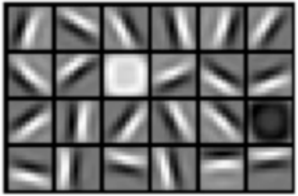
$$\underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Horizontal Filter}}$$





Hierarchical Feature Detection

Feature Map as an Indicator: The output feature map highlights detected patterns, with higher values indicating matched regions.



- **Early Layers**: Detect basic elements like edges and textures, forming the foundation for more complex patterns.
- **Middle Layers**: Combine edges into shapes (e.g., circles, squares) by recognizing the arrangement of basic features.
- **Deeper Layers**: Recognize object parts by detecting combinations of shapes and features.
- **Final Layers**: Detect entire objects by assembling recognized parts, outputting a classification or region of interest.

Lee, et al. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." ICML 2009.

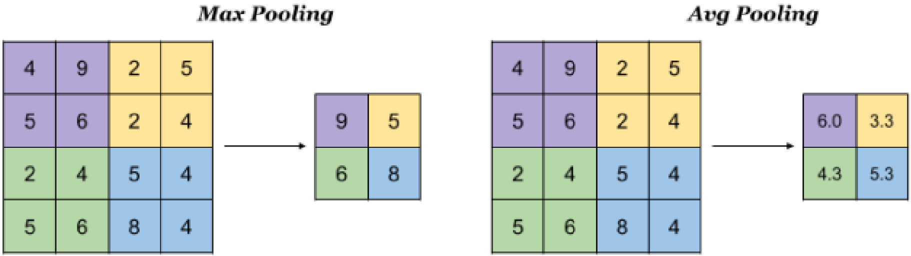
Summary of Convolutional Neural Networks

- **Challenges:** High dimensionality, translation invariance, and spatial structure
- **Filters:** Small, **trainable** matrices that detect features in the input data.
- **Convolution Operation:** A **locally linear transformation** that creates a feature map, emphasizing regions where the filter matches the pattern.
- **Padding and Stride:** Methods for controlling feature map size, preserving spatial dimensions, and improving *computational efficiency*.
- **Convolution Over Volumes:** Designed to process multi-channel inputs like RGB images with filters that **match** each channel.
- **Multiple Filters:** A single convolutional layer can use **multiple** filters to detect various features simultaneously.
- **Weight Sharing and Sparsity:** Neurons in CNNs **share** weights across locations, with each output relying on a **small, localized** input region.
- **Hierarchical Feature Detection:** Early layers capture basic features (like edges), which later layers combine into higher-level features.

Outline

Pooling

Define: The pooling layer in CNNs reduces spatial dimensions of feature maps through downsampling, commonly using max or average pooling operations.



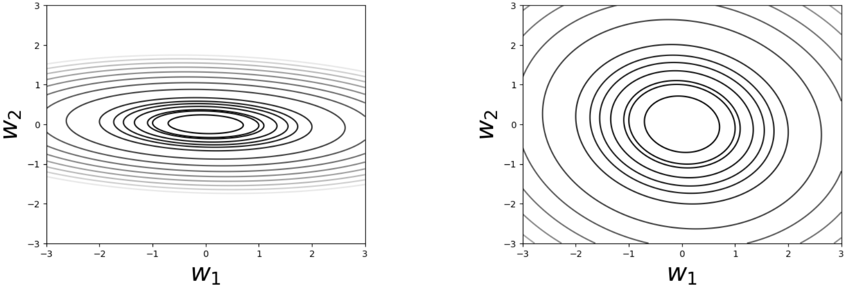
- Pooling helps reduce the computational load
- It also enhances **robustness** by making the network less sensitive to small spatial variations.
- Common hyperparameters: pool size f and stride s, typically $f = s = 2$.

Recap: Input Normalization

- Normalize the inputs using **training set**:

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i, \qquad \boldsymbol{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu})^2, \qquad \bar{\boldsymbol{x}}_i = (\boldsymbol{x}_i - \boldsymbol{\mu})/\boldsymbol{\sigma},$$

  where all operations are taken element-wise.
- Consider a binary classification problem using linear model: $f_{\boldsymbol{\theta}}(x) = \boldsymbol{w}^\top \boldsymbol{x} = w_1 x_1 + w_2 x_2$
  - if $x_1 = \mathcal{O}(100)$ and $x_2 = \mathcal{O}(1)$, to have output $f_{\boldsymbol{\theta}} = \mathcal{O}(1)$, we must have $w_1 = \mathcal{O}\left(\frac{1}{100}\right)$ and $w_2 = \mathcal{O}(1)$.
  - After normalization, $\bar{x}_1 = \mathcal{O}(1)$ and $\bar{x_2} = \mathcal{O}(1)$, so we have $w_1 = \mathcal{O}(1)\,1$ and $w_2 = \mathcal{O}(1)$.



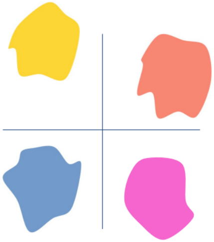
- At test time, apply $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ from **training** to test set.

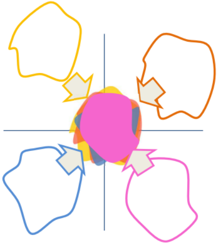
Batch Normalization

- Given an input $\boldsymbol{x}$, the forward propagation in DNNs:

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1}, \quad \boldsymbol{a}^{\ell} = \phi(\boldsymbol{z}^{\ell}) \quad \forall \ell \in [L].$$

  where $\boldsymbol{x}^0 = \boldsymbol{x}$ is the input image.
- We can apply the same normalization from the input layer to each hidden layer to speed up the training.
- Additionally, as we train DNNs using mini-batch rather than full batch, **internal covariance shift** in *mini-batches*. Hence, the normalization is applied on the mini-batch rather the full batches.





Batch Normalization During Training

Given a mini-batch $\{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^b\}$ for a hidden layer:

- Normalize the **pre-activation** to mean zero and variance one:

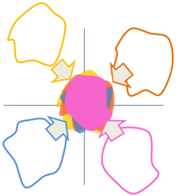
$$\boldsymbol{\mu} = \frac{1}{b}\sum_{i=1}^{b} \boldsymbol{z}^i, \qquad \boldsymbol{\sigma}^2 = \frac{1}{b}\sum_{i=1}^{b} (\boldsymbol{z}^i - \boldsymbol{\mu})^2, \qquad \boldsymbol{z}^i_{\text{norm}} = \frac{\boldsymbol{z}^i - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \varepsilon}}$$

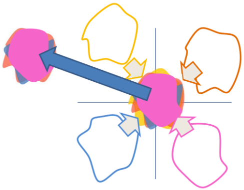
  where $\varepsilon > 0$ ensures numerical stability.

- Re-scale and shift using learnable parameters:

$$\hat{\boldsymbol{z}}^i = \boldsymbol{\gamma} \boldsymbol{z}^i_{\text{norm}} + \boldsymbol{\beta}$$

  where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are **learnable** parameters.





- $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ enable identity transformation, allowing flexibility:

$$\boldsymbol{\gamma} = \sqrt{\boldsymbol{\sigma}^2 + \varepsilon}, \qquad \boldsymbol{\beta} = \boldsymbol{\mu}$$

Batch Normalization at Test Time

- **Training Phase:** For each mini-batch, compute batch statistics and update the normalized output:

$$\boldsymbol{\mu}_{\text{batch}} = \frac{1}{b}\sum_{i=1}^{b} \boldsymbol{z}^i, \quad \boldsymbol{\sigma}^2_{\text{batch}} = \frac{1}{b}\sum_{i=1}^{b}(\boldsymbol{z}^i - \boldsymbol{\mu}_{\text{batch}})^2, \quad \boldsymbol{z}^i_{\text{norm}} = \frac{\boldsymbol{z}^i - \boldsymbol{\mu}_{\text{batch}}}{\sqrt{\boldsymbol{\sigma}^2_{\text{batch}} + \varepsilon}}, \quad \hat{\boldsymbol{z}}^i = \boldsymbol{\gamma}\boldsymbol{z}^i_{\text{norm}} + \boldsymbol{\beta}$$

- **Running Statistics:** After each mini-batch, update the running mean and variance:

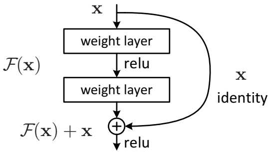
$$\begin{aligned}\boldsymbol{\mu}_{\text{run}} &= (1-\alpha)\boldsymbol{\mu}_{\text{run}} + \alpha\boldsymbol{\mu}_{\text{batch}} \\ \boldsymbol{\sigma}^2_{\text{run}} &= (1-\alpha)\boldsymbol{\sigma}^2_{\text{run}} + \alpha\boldsymbol{\sigma}^2_{\text{batch}}\end{aligned}$$

- **Test Phase:** Normalize using running statistics and apply scale and shift:

$$\boldsymbol{z}_{\text{test}} \leftarrow \frac{\boldsymbol{z}_{\text{test}} - \boldsymbol{\mu}_{\text{run}}}{\sqrt{\boldsymbol{\sigma}^2_{\text{run}} + \varepsilon}}, \quad \hat{\boldsymbol{z}}_{\text{test}} \leftarrow \boldsymbol{\gamma}\boldsymbol{z}_{\text{test}} + \boldsymbol{\beta}$$

Skip Connections

Define: A skip connection is a shortcut in a DNN that adds the input directly to the output.



- In MLPs, forward propagation without skip connections (omitting biases):

$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1}) \approx \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1} \approx \boldsymbol{W}^{\ell}\cdots\boldsymbol{W}^{1}\boldsymbol{x}^{0} = \mathcal{O}\left(a^{\ell}\right)$$

  This results in an **exponential growth or decay** of information.

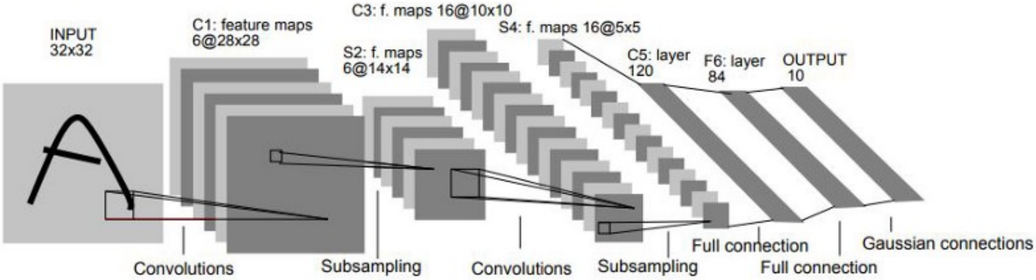
- With skip connections, the propagation becomes:

$$\boldsymbol{x}^{\ell} = \phi(\boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1}) + \boldsymbol{x}^{\ell-1} = \sum_{i=0}^{\ell}\boldsymbol{W}^{i}\phi(\boldsymbol{x}^{i}) = \mathcal{O}\left(\ell\right)$$

  Here, **linear growth** of information is achieved, stabilizing the learning process.

Outline

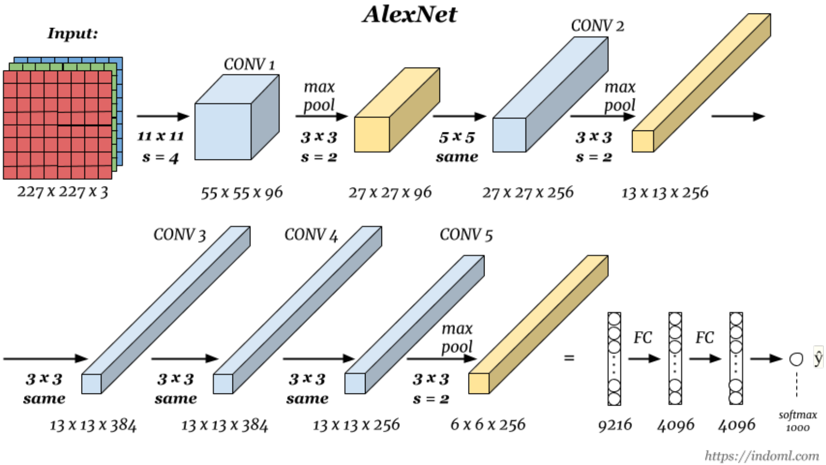
LeNet-5



- It has totally 5 weight layers: 2 Convolutional layers and 3 fully connected (FC) layers
- Sigmoid and $\tanh$ activations
- Average pooling
- Number of parameters: ~ 60 thousands.
- MNIST dataset: ~ 60 thousands.

LeCun, Y., et al. (1998). "Gradient-based learning applied to document recognition." Proc. of the IEEE

AlexNet

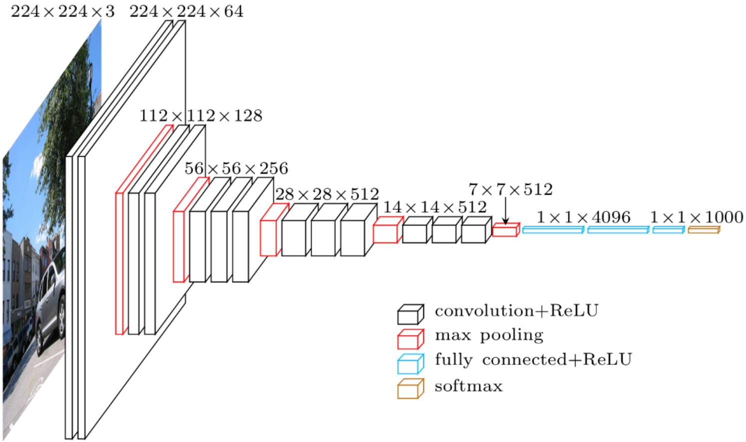


- 5 convolutional and 3 FC.
- ReLU activation and max pooling layers.
- Dropout regularization in FC layers.

- Number of parameters: $\sim$ 63 million.
- ImageNet dataset: $\sim$ 1.2 million images.

Krizhevsky, Alex, et al. "Imagenet classification with deep convolutional neural networks." NeurIPS 2012.

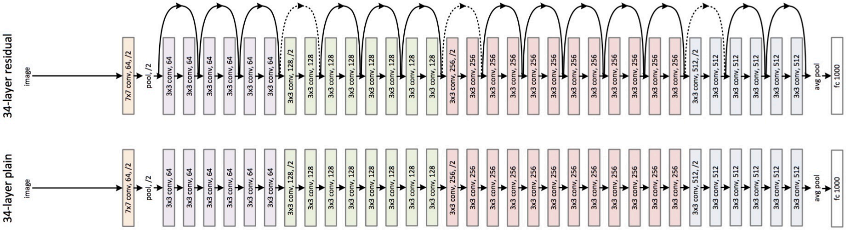
VGG-16

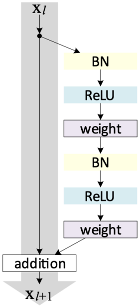


- 13 convolutional and 3 FC.
- *Unified* convolution and max-pooling setup: $f = 3$, $s = 1$, and "same"; $f = 2$ and $s = 2$
- ~ 138 million parameters trained on ImageNet

Simonyan, K., & Zisserman, A. "Very deep convolutional networks for large-scale image recognition." ICLR 2015.

ResNet-34





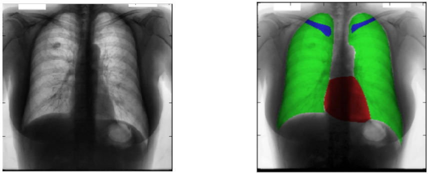
- Batch normalization and skip connections applied to pre-activation.
- ~ 11.7 million parameters trained on ImageNet.

He, Kaiming, et al. "Deep residual learning for image recognition." CVPR 2016.

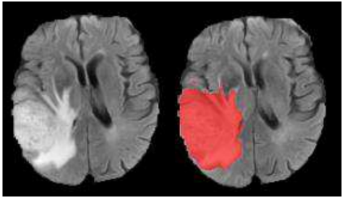
Outline

Semantic Segmentation with U-Net

Successful Applications:



Chest X-Ray



Brain MRI

Novikov, et al. "Fully convolutional architectures for multiclass segmentation in chest radiographs." IEEE Tran Med Img 2018
Dong, et al. "Automatic brain tumor detection and segmentation using U-Net based fully convolutional networks.", MIUA2017

Semantic Labeling

Per-Pixel Class Labeling:

Input

segmented

1:Person
2:Purse
3:Plants/Grass
4:Sidewalk
5:Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Semantic Labels

- Assign a class label to every pixel in the image.
- Output is an image of the same dimensions as the input.

Gao, et al. "SFSM: sensitive feature selection module for image semantic segmentation." Multimed. Tools Appl. 2023

Transpose Convolution

Transpose Convolution: A transpose convolution (or a **deconvolution** or **up-sampling convolution**) is an operation that applies a filter to input data in a way that expands its spatial dimensions.

$$\underbrace{\begin{bmatrix} 3 & 0 \\ 1 & 5 \end{bmatrix}}_{\textbf{input } 2 \times 2} * \underbrace{\begin{bmatrix} 2 & 7 & 4 \\ 3 & 1 & 7 \\ 4 & 2 & 1 \end{bmatrix}}_{\textbf{filter } 3 \times 3} = \underbrace{\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}}_{\textbf{feature map } 4 \times 4}$$

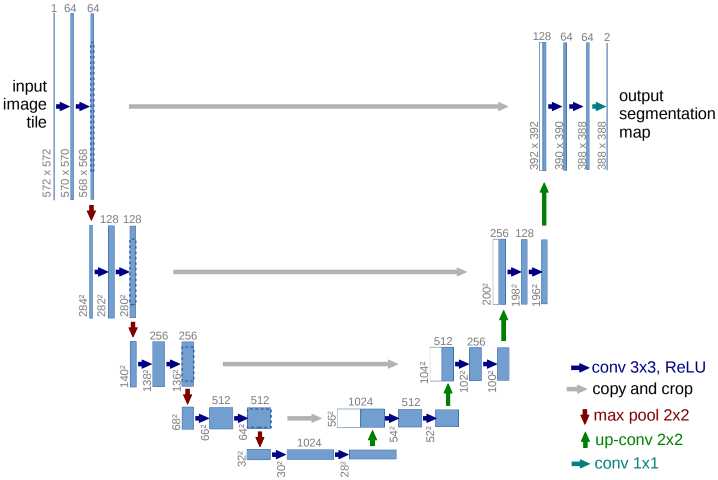
Deconvolution

Deconvolution: A deconvolution (or a **transpose convolution** or **up-sampling convolution**) is an operation that applies a filter to input data in a way that expands its spatial dimensions.

$$\underbrace{\begin{bmatrix} 3 & 0 \\ 1 & 5 \end{bmatrix}}_{\textbf{input } 2 \times 2} * \underbrace{\begin{bmatrix} 2 & 7 & 4 \\ 3 & 1 & 7 \\ 4 & 2 & 1 \end{bmatrix}}_{\textbf{filter } 3 \times 3} = \underbrace{\begin{bmatrix} 6 & 21 & 12 & 0 \\ 11 & 20 & 60 & 20 \\ 15 & 23 & 24 & 35 \\ 4 & 12 & 11 & 5 \end{bmatrix}}_{\textbf{feature map } 4 \times 4}$$

- The stride can be more than 1
- Padding is reversed by discarding boundary pixels.
- For overlaps, use averaging or summation.
- The filter represents patterns, with the input indicating where these patterns are detected.
- In unmax pooling, either duplicate pixels in the output or place the maximum value pixel while setting others to zero.

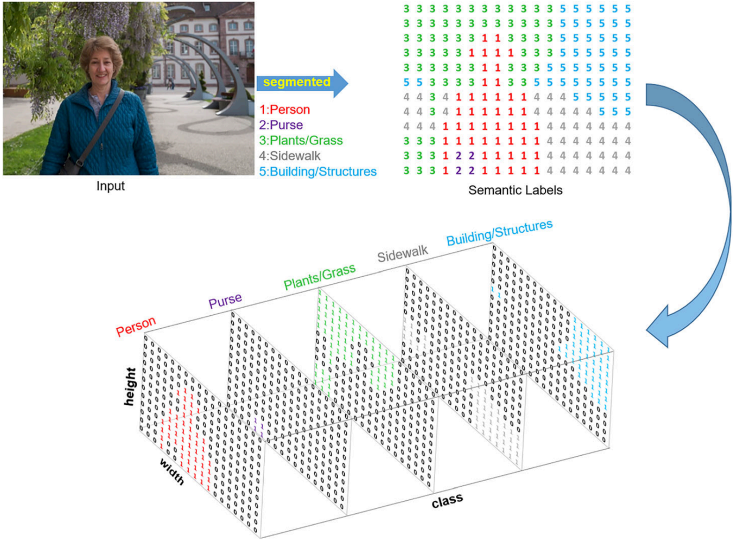
U-Net Architecture



- With skip connection, U-Net combines (or concatenates) **high-level abstract features** (from deeper layers) and **spatial details** (from earlier layers).

Ronneberger, et al. "U-net: Convolutional networks for biomedical image segmentation." MICCAI 2015

U-Net Output



Gao, et al. "SFSM: sensitive feature selection module for image semantic segmentation." Multimed. Tools Appl. 2023