Introduction to Deep Learning
Complementary Course in Statistical Learning and Applications — Master 1 AM UGA

Razan MHANNA

April 2025

Introduction to Deep Learning

- Deep learning is an active area of research in machine learning and AI.
- The cornerstone of deep learning is the **neural network**.
- Neural networks gained popularity in the 1980s but declined due to competing methods like SVMs and boosting.
- Revival post-2010 due to larger datasets and better architectures.

The Evolution of Neural Networks

- 1980s: Initial rise with significant excitement and NeurIPS conferences.

The Evolution of Neural Networks

- 1980s: Initial rise with significant excitement and NeurIPS conferences.
- 1990s-2000s: Decline due to complexity and competition.

The Evolution of Neural Networks

- 1980s: Initial rise with significant excitement and NeurIPS conferences.
- 1990s-2000s: Decline due to complexity and competition.
- Post-2010: Revival as **deep learning**, excelling in image, video, speech, and text processing.

Specialized Neural Networks

- **Convolutional Neural Networks (CNNs):** Used for image classification.
- **Recurrent Neural Networks (RNNs):** Applied in time series and sequence modeling.
- **Transformers:** State-of-the-art models for NLP tasks.
- ...

Single Layer Neural Networks

Definition

A neural network takes an input vector of p variables $X = (X_1, X_2, ..., X_p)$ and builds a nonlinear function $f(X)$ to predict the response Y.

- Neural networks differ from other models due to their structured approach.

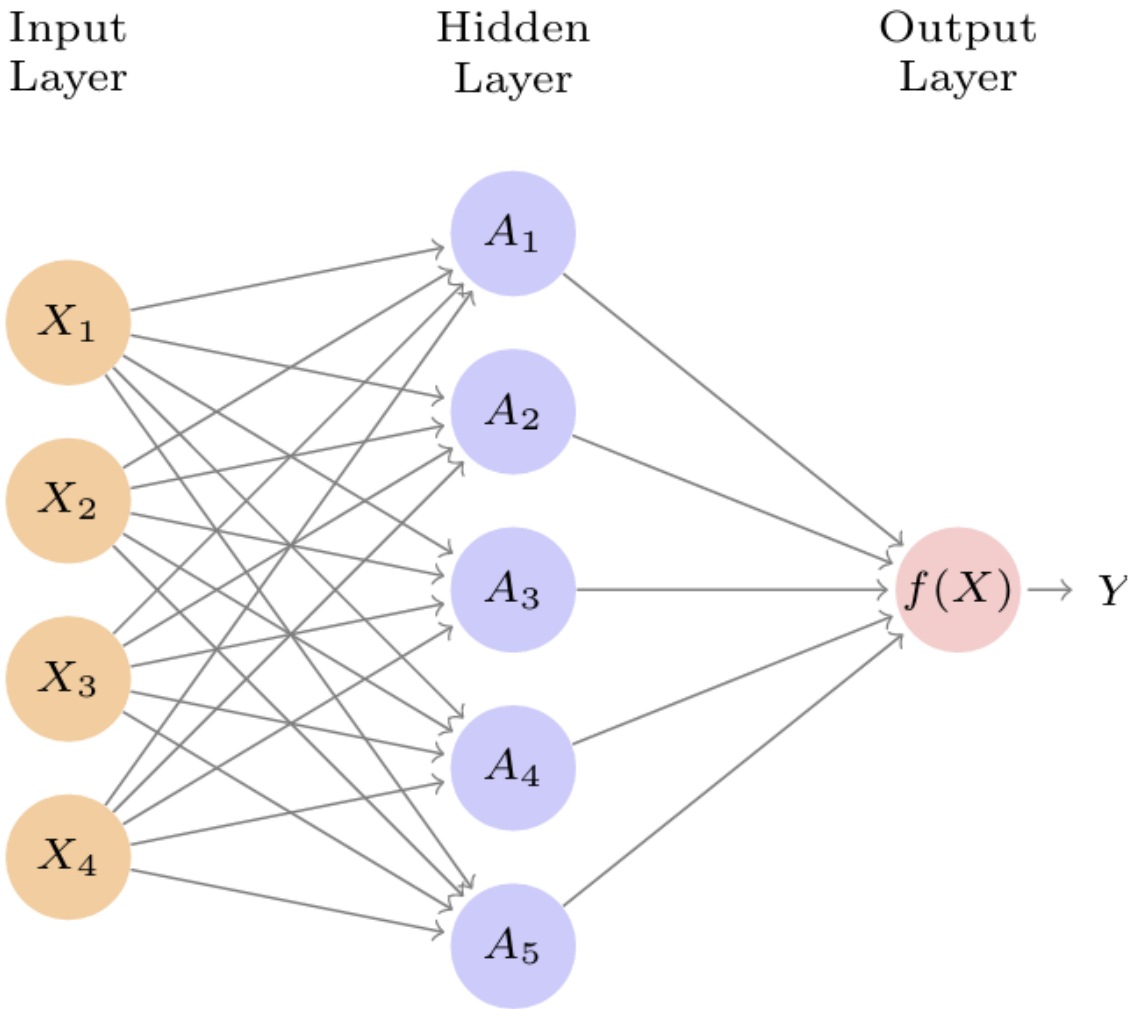
Single Layer Neural Networks

Definition

A neural network takes an input vector of p variables $X = (X_1, X_2, ..., X_p)$ and builds a nonlinear function $f(X)$ to predict the response Y.

- Neural networks differ from other models due to their structured approach.
- A simple feed-forward neural network consists of:
  - **Input layer**: Takes p predictors.
  - **Hidden layer**: Consists of K hidden units, applying transformations.
  - **Output layer**: Combines transformed features to predict Y.

Example of a simple feed forward neural network

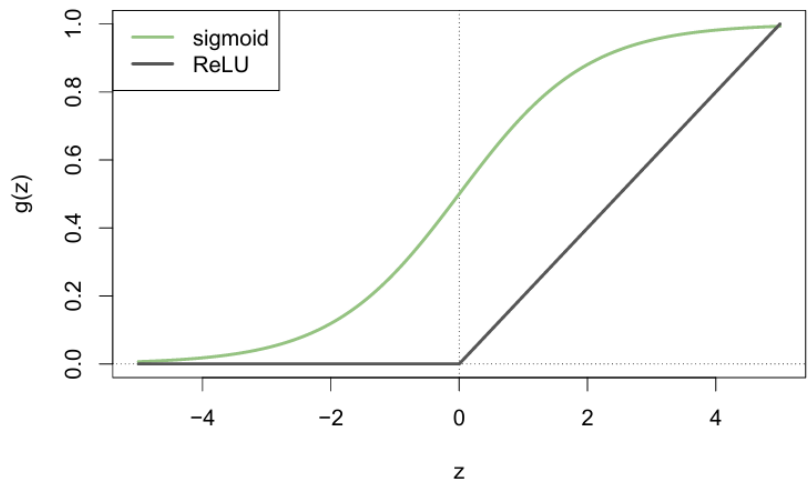


Activation Functions in Neural Networks

Definition

An activation function $g(z)$ is applied to hidden units to introduce non-linearity, allowing the model to learn complex relationships.

- Common activation functions:
  - **Sigmoid**: $g(z) = \frac{1}{1+e^{-z}}$, commonly used in early networks.
  - **ReLU (Rectified Linear Unit)**: $g(z) = \max(0, z)$, preferred in modern deep learning for efficiency.
- Non-linearity in activation functions prevents the network from collapsing into a simple linear model.



Example: Neural Network with Two Inputs

Example Setup

Consider a simple case with $p = 2$ input variables $X = (X_1, X_2)$ and $K = 2$ hidden units with activation function $g(z) = z^2$.

- Parameters:
  - $\beta_0 = 0, \beta_1 = \frac{1}{4}, \beta_2 = \frac{1}{4}$
  - $w_{10} = 0, w_{11} = 1, w_{12} = 1$
  - $w_{20} = 0, w_{21} = 1, w_{22} = -1$
- Hidden unit activations:
  - $h_1(X) = (0 + X_1 + X_2)^2$
  - $h_2(X) = (0 + X_1 - X_2)^2$

Computation of Output

Final Function

The final function is computed as:

$$f(X) = \beta_0 + \beta_1 h_1(X) + \beta_2 h_2(X) \tag{1}$$

Expanding:

Computation of Output

Final Function

The final function is computed as:

$$f(X) = \beta_0 + \beta_1 h_1(X) + \beta_2 h_2(X) \tag{1}$$

Expanding:

$$f(X) = \frac{1}{4}(X_1 + X_2)^2 - \frac{1}{4}(X_1 - X_2)^2 \tag{2}$$

Which simplifies to:

$$f(X) = X_1 X_2 \tag{3}$$

- Nonlinear transformations can model interactions between inputs.
- Activation functions like ReLU or sigmoid avoid restricting the model to polynomials.

Multilayer Neural Networks

Definition

Modern neural networks typically have multiple hidden layers, each with several units, improving learning efficiency.

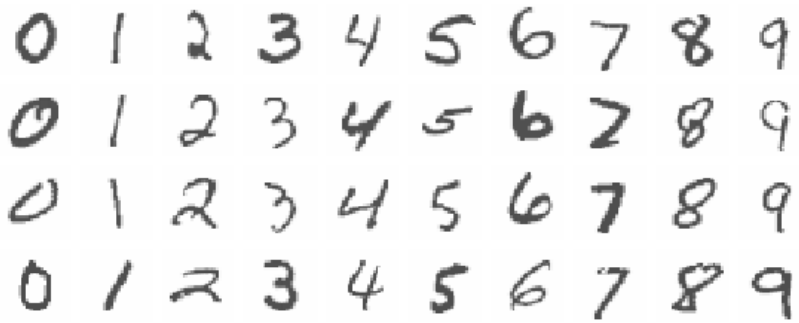
- A single hidden layer with enough units can approximate most functions, but multiple layers simplify learning.

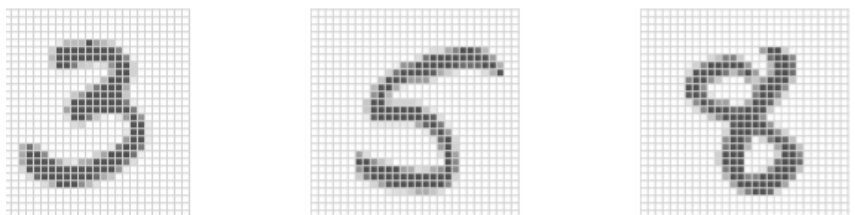
Multilayer Neural Networks

Definition

Modern neural networks typically have multiple hidden layers, each with several units, improving learning efficiency.

- A single hidden layer with enough units can approximate most functions, but multiple layers simplify learning.
- Example: **MNIST Handwritten Digit Classification**
  - Dataset: 60,000 training images, 10,000 test images.
  - Each image has $28 \times 28 = 784$ grayscale pixels.
  - Output: One-hot encoded class labels (digits 0-9).



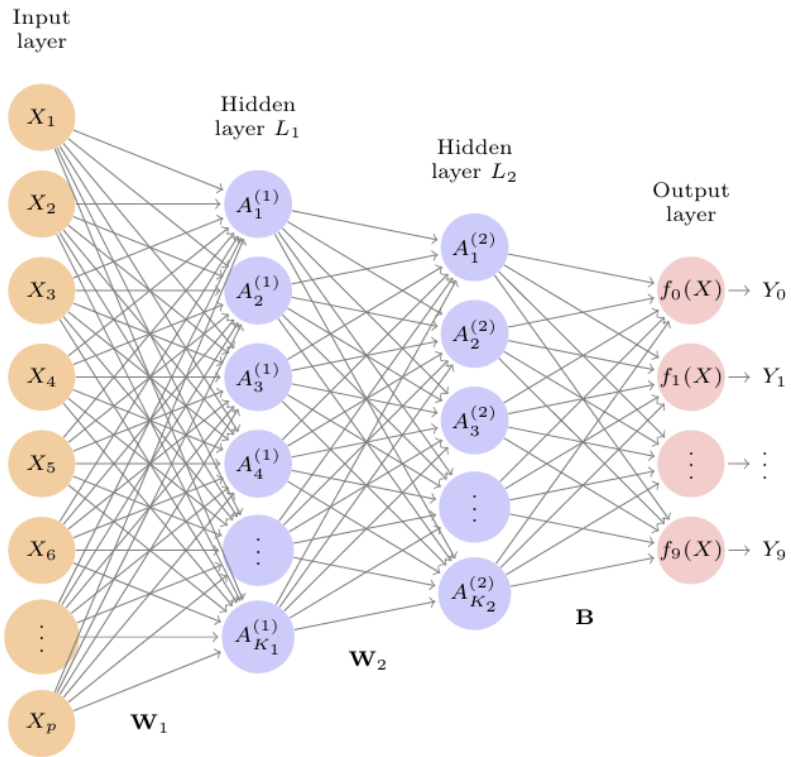


Neural Network Architecture for MNIST

- Input Layer: 784 units (one per pixel).
- Two Hidden Layers:
  - $L1$ - 256 units
  - $L2$ - 128 units
- Output Layer: 10 units (one per digit class).
- Total parameters: 235,146 (weights and biases).

Neural Network Architecture for MNIST

- Input Layer: 784 units (one per pixel).
- Two Hidden Layers:
  - $L1$ - 256 units
  - $L2$ - 128 units
- Output Layer: 10 units (one per digit class).
- Total parameters: 235,146 (weights and biases).



Training and Loss Function

- Each hidden layer computes activations:

$$A_k^{(1)} = g\left(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j\right)$$

$$A_\ell^{(2)} = g\left(w_{\ell 0}^{(2)} + \sum_{k=1}^{K_1} w_{\ell k}^{(2)} A_k^{(1)}\right)$$

- Output layer: 10 different linear models

$$f_m(X) = z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}$$

- We use softmax for probability estimation: [1]

$$f_m(X) = \frac{e^{Z_m}}{\sum_{m=0}^{9} e^{Z_m}}$$

[1] Loss function: Cross-entropy minimization $-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im} \log f_m(x_i)$ For a quantitative response, we would instead minimize the squared-error loss.

Performance Comparison

- Regularization helps avoid overfitting:
  - Ridge Regularization: 2.3% test error.
  - Dropout Regularization: 1.8% test error.
- Comparison with linear models:
  - Multinomial Logistic Regression: 7.2%
  - Linear Discriminant Analysis: 12.7%

Method	**Test Error**
Neural Network + Ridge	2.3%
Neural Network + Dropout	1.8%
Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

Table: Test error rates on MNIST dataset.

Introduction to Convolutional Neural Networks

- Neural networks gained popularity around 2010 due to breakthroughs in image classification.
- Large labeled image datasets enabled deep learning advances.

Introduction to Convolutional Neural Networks

- Neural networks gained popularity around 2010 due to breakthroughs in image classification.
- Large labeled image datasets enabled deep learning advances.
- Example: **CIFAR-100** – A dataset with 60,000 images across 100 classes.

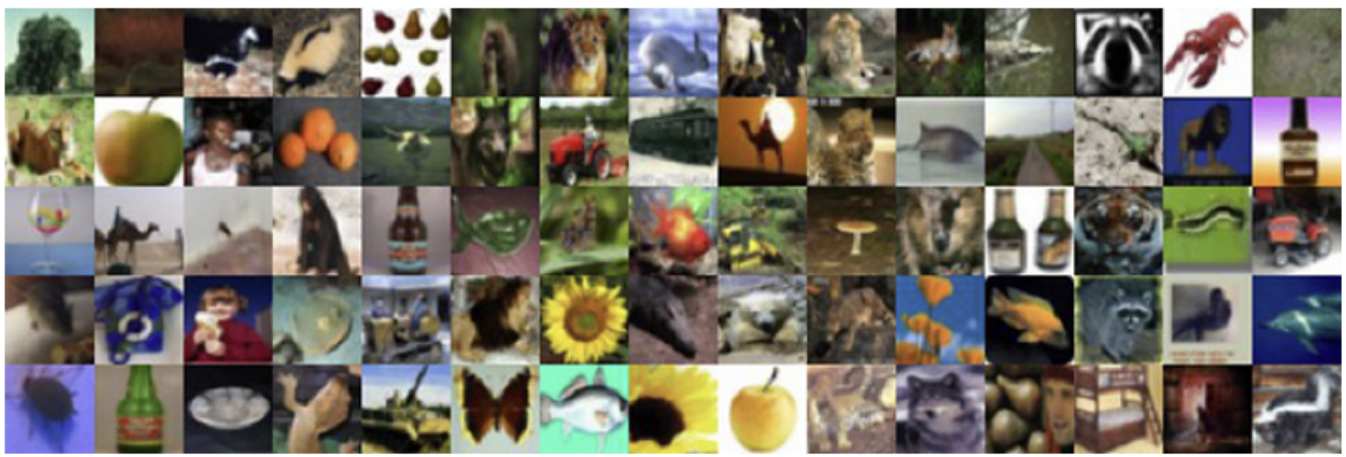


Image Representation in CNNs

CIFAR100:

- Images are stored as 3D arrays called **feature maps**.
- Each image has:
  - Height (H) $\times$ Width (W) (e.g., 32 $\times$ 32 pixels).
  - Color channels (C) – Red, Green, Blue (RGB).
- Training set: 50,000 images, Test set: 10,000 images.

Image Representation in CNNs

CIFAR100:

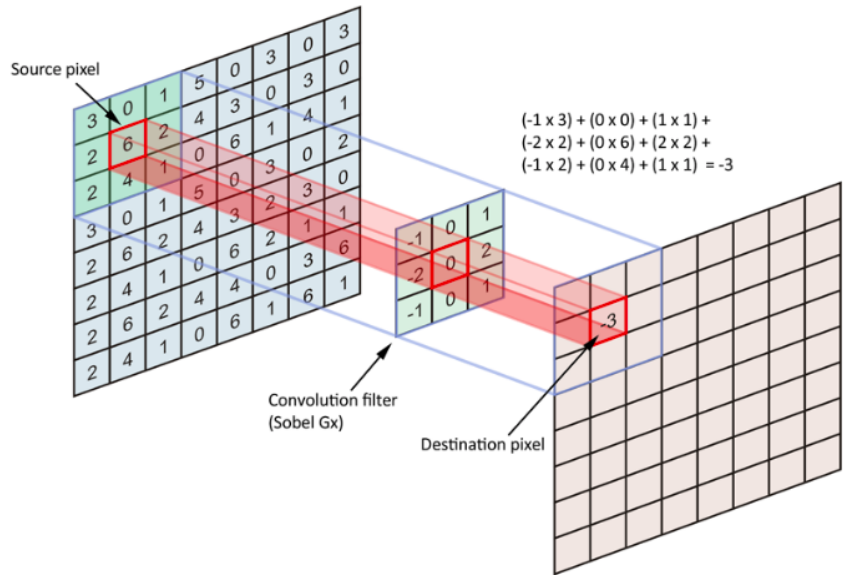
- Images are stored as 3D arrays called **feature maps**.
- Each image has:
  - Height (H) $\times$ Width (W) (e.g., 32 $\times$ 32 pixels).
  - Color channels (C) – Red, Green, Blue (RGB).
- Training set: 50,000 images, Test set: 10,000 images.

How CNNs Work:

- CNNs process images hierarchically:
  - Identify low-level features: edges, colors.
  - Combine them into complex patterns: shapes, textures.
  - Recognize objects by analyzing feature relationships.

Convolution Layers

- The core component of CNNs is the **convolution layer**.
- Uses small templates called **filters** or **kernels** to scan images.
- Operation: **Convolution** – multiplies kernel values with pixel values, then sums.
- Purpose: Detect features like edges, textures, and patterns.



Example of Convolution

Original Image (4×3 pixels):

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}$$

2×2 Convolution Filter:

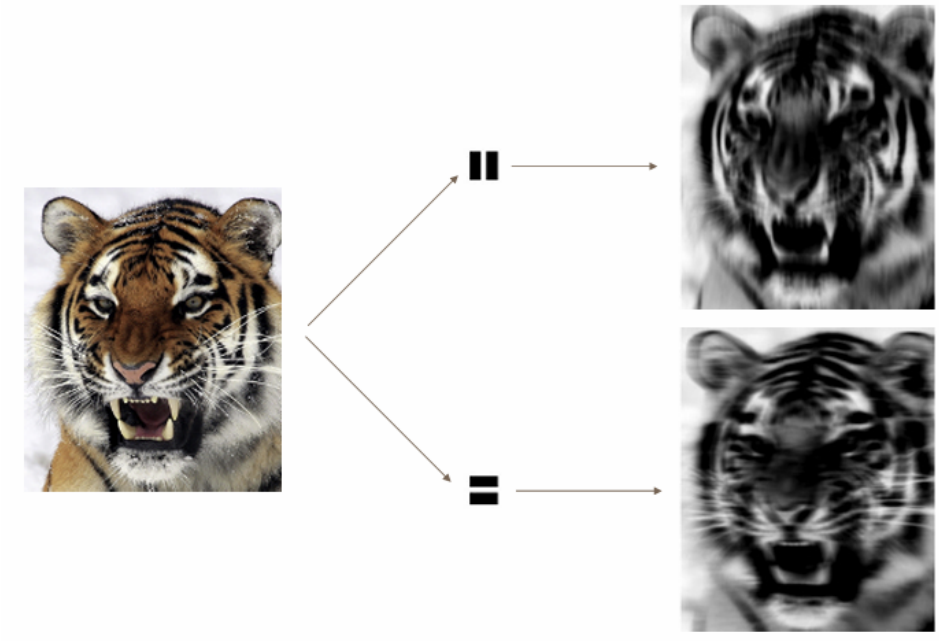
$$\begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

Convolved Image:

$$\begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

Edge Detection filters using Convolution

- Applying specific filters enhances particular features.
- Example: Detecting horizontal and vertical edges.



- In image preprocessing, we use predefined filter.
- In CNN, filters are learned instead!

CNN Feature Maps and Filters

- The input image has 3 color channels (RGB), forming a 3D feature map each a 32×32 matrix.
- A convolution filter also has 3 channels, each of size 3×3.
- The results of the convolution are summed to form a two-dimensional output feature map.
- Using K different filters in the first hidden layer produces K output feature maps, forming a new three-dimensional feature map with k channels.
- This three-dimensional feature map is structured similarly to hidden layers in traditional neural networks.

Pooling Layers

- Reduce the spatial dimensions while preserving important features.
- Types:
  - Max Pooling: Keeps the highest value in each region.
  - Average Pooling: Takes the average value in each region.

Here is a simple example of max pooling:

$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}.$$

Pooling Layers

- Reduce the spatial dimensions while preserving important features.
- Types:
  - Max Pooling: Keeps the highest value in each region.
  - Average Pooling: Takes the average value in each region.

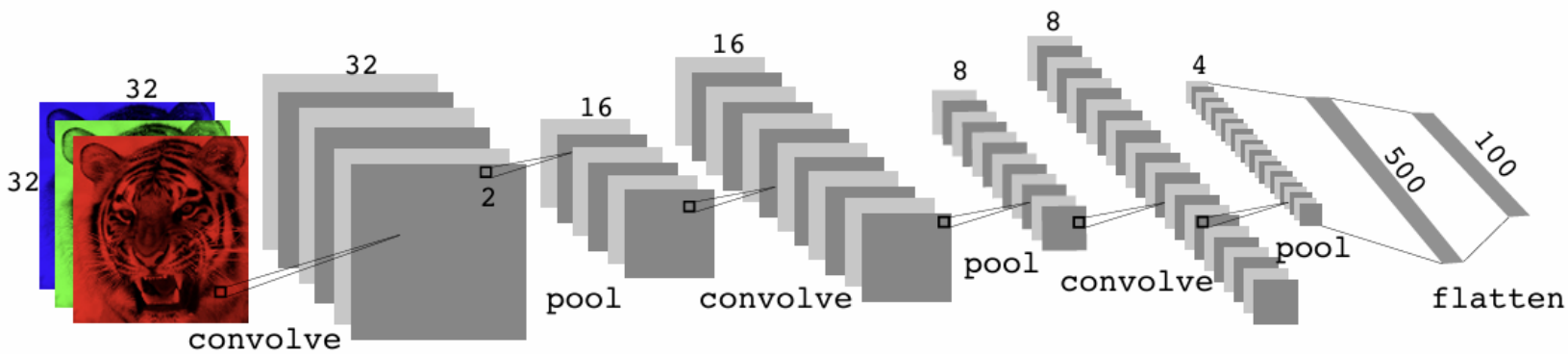
Here is a simple example of max pooling:

$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}.$$

- Max Pooling uses a 2×2 block to take the maximum value within each block.
- This reduces the image size by a factor of 2 and provides some location invariance.

CNN Architecture

- A typical CNN architecture includes many convolution and pooling layers.
- Each convolution layer applies multiple filters, and the output is passed through a pooling layer.
- After several layers, the final layers are typically fully connected.



Subsequent Convolution Layers

- Each subsequent convolution layer takes as input the 3D feature map from the previous layer.
- The convolution filter in each layer has as many channels as the feature map from the previous layer.
- As the feature map reduces in size after each pooling layer, the number of filters is often increased in the next convolution layer.
- Sometimes several convolution layers are used before a pooling layer.

Flattening and Fully Connected Layers

- After multiple convolution and pooling layers, the 3D feature maps are flattened.
- The pixels are treated as separate units and fed into one or more fully-connected layers.
- The output layer is typically a softmax activation function, providing class predictions (e.g., for CIFAR-100 with 100 classes).

Summary of CNN

- CNNs are specialized for image classification tasks.
- Use convolution and pooling layers to extract hierarchical features.
- Modern architectures stack multiple layers for improved accuracy.
- Widely used in applications such as object detection, facial recognition, and medical imaging.

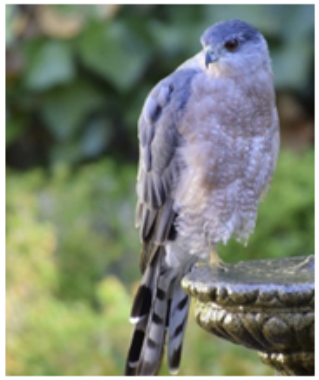
Data Augmentation

- Data augmentation involves replicating training images and applying random distortions, such as zoom, shifts, shear, rotations, and flips.
- This helps protect against overfitting and acts as a form of regularization.
- Augmentation builds a cloud of images around each original image with the same label.

Pretrained Classifier - ResNet50

- ResNet50 is a convolutional neural network trained on the ImageNet dataset, which consists of millions of images across numerous categories.
- ResNet50 can be used for classifying new images by applying the pretrained network.
- The example bellow shows how ResNet50 classifies images like hawks and birds.
- The classifier performs well on certain images, but can confuse when details change (e.g., zooming out).

Classification example of six photographs using the resnet50



flamingo		Cooper's hawk		Cooper's hawk	
flamingo	0.83	kite	0.60	fountain	0.35
spoonbill	0.17	great grey owl	0.09	nail	0.12
white stork	0.00	robin	0.06	hook	0.07

Lhasa Apso		cat		Cape weaver	
Tibetan terrier	0.56	Old English sheepdog	0.82	jacamar	0.28
Lhasa	0.32	Shih-Tzu	0.04	macaw	0.12
cocker spaniel	0.03	Persian cat	0.04	robin	0.12

Weight Freezing and Transfer Learning

- In transfer learning, pretrained networks like ResNet50 can be adapted to new problems.
- **Weight freezing** involves keeping the pretrained filters frozen and only training the last layers.
- This requires less data for training and leverages the features learned from large datasets like ImageNet.

Weight Freezing and Transfer Learning

```
import torchvision.models as models
import torch.nn as nn

# Load pre-trained model
model = models.resnet50(pretrained=True)

# Freeze all layers
for param in model.parameters():
    param.requires_grad = False

# Replace the classifier layer
model.fc = nn.Linear(model.fc.in_features, 2) \\
# Now only train model.fc
```