Session 4

Regularization and Convolutional Neural Networks

Deep Learning course - SKEMA 2025

Mastère Spécialisé® Chef de Projet Intelligence Artificielle

Salem Lahlou

Some figures adapted from 3Blue1Brown (YouTube)

From Shallow to Deep Networks

Shallow Neural Networks:

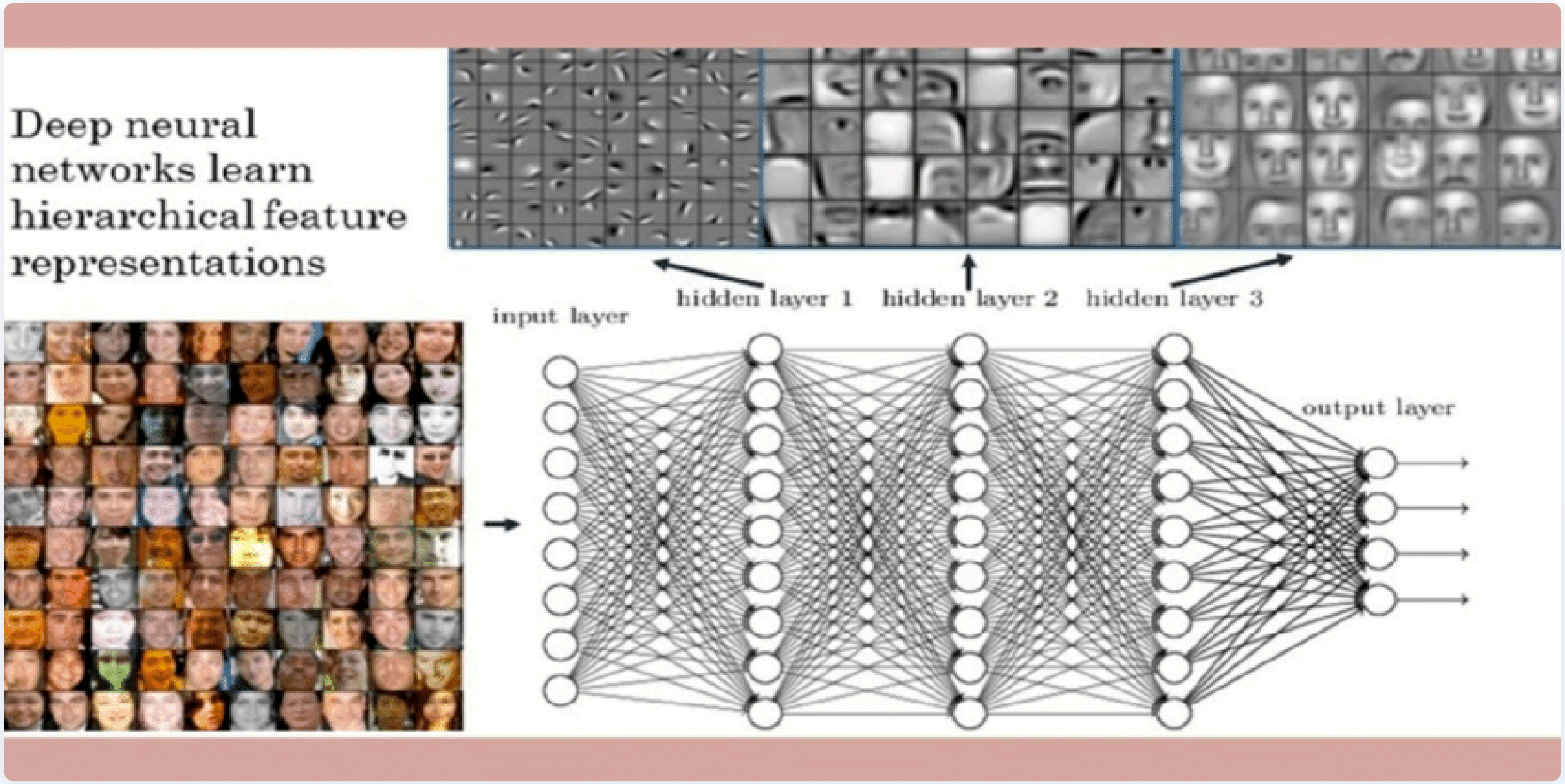
- One or few hidden layers
- Limited representation power

Deep Neural Networks:

- Many hidden layers
- Increased capacity to learn complex patterns

The Power of Representation Learning

Key Insight: Each layer learns increasingly abstract features



Example in image recognition:

- First layers: Edges and simple shapes
- Middle layers: Parts and textures
- Deep layers: Complex objects and concepts

Overfitting: The Fundamental Challenge

What is overfitting?

- Model performs well on training data
- Model performs poorly on new, unseen data
- Model learns noise instead of true patterns

Signs of overfitting:

- Large gap between training and validation error
- Model complexity increases but validation performance worsens
- Perfect or near-perfect training accuracy

Why it happens:

- Too many parameters relative to data points
- Training too long
- Insufficient regularization
- Noisy training data

Comprehensive Strategies to Combat Overfitting

1. Data-based approaches:

- More training data
- Data augmentation
- Noise injection
- Feature selection

2. Model complexity reduction:

- Simpler models (fewer layers/neurons)
- Early stopping
- Pruning

3. Regularization techniques:

- L1 regularization (Lasso): $\lambda \sum_{i=1}^{p} |\theta_i|$
- L2 regularization (Ridge): $\lambda \sum_{i=1}^{p} \theta_i^2$
- Dropout: Randomly "drop" neurons during training
- Batch normalization: Normalize layer inputs

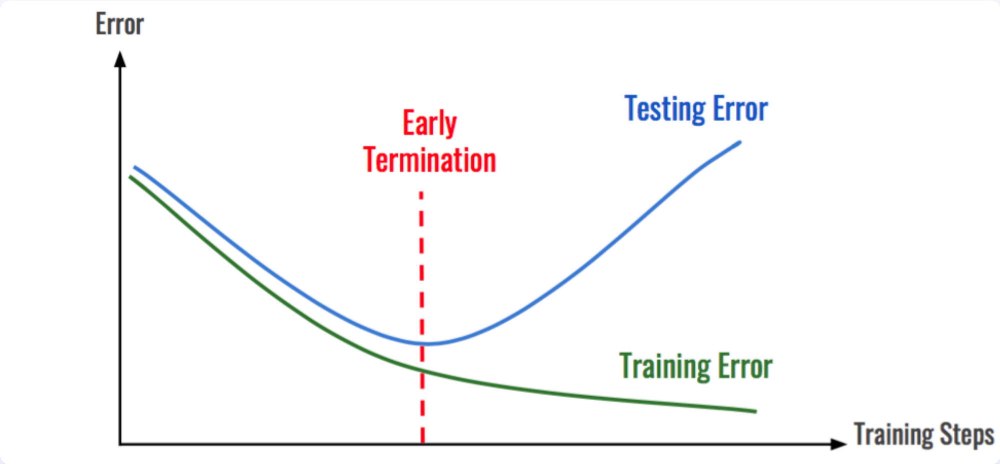
Early Stopping: A Simple but Effective Technique

Concept:

- Monitor validation error during training
- Stop when validation error starts to increase
- Take model parameters from best validation point

Implementation:

- Track validation metrics after each epoch
- Save model whenever validation improves
- Use patience parameter to allow for fluctuations



Source: Analytics Vidhya

Specialized Architectures: Introduction to CNNs

Specialized neural networks for specific data types:

- Convolutional Neural Networks (CNNs) for images and spatial data
- Recurrent Neural Networks (RNNs) for sequential data
- Transformers for sequential data with attention mechanisms

Today we'll focus on CNNs:

- Inspired by visual processing in the brain
- Excellent for image recognition and computer vision
- Leverages spatial structure in data

The Challenge of Image Recognition

Why are images difficult for standard neural networks?

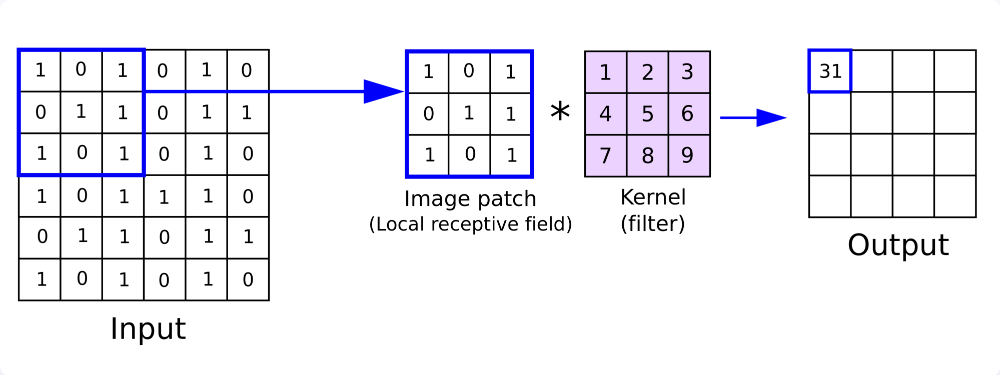
- High dimensionality (millions of pixels)
- Spatial relationships matter
- Same object can appear in different positions
- Lighting, angle, and scale variations

Example:

- A 256x256 RGB image has 196,608 pixels ($256 \times 256 \times 3$)
- A fully connected first layer would need millions of parameters

CNNs: Inspired by the Visual Cortex

- Neurons in visual cortex respond to specific regions of visual field
- Different neurons detect different features (edges, shapes, etc.)
- CNNs mimic this structure with specialized layers



Source: Anh Reynolds

Advantages:

- Drastically reduces parameters compared to fully connected
- Preserves spatial relationships
- Translation invariance

CNN Components: Convolutional Layers

Key innovation: Parameter sharing and local connectivity

How convolution works:

- Filters (kernels) slide across the input image
- Each filter performs the same operation at each position
- Captures local patterns regardless of position

Mathematical representation:

- For a filter W applied to image region X:
- Output = $\sum_{i,j} W_{i,j} \cdot X_{i,j} + b$
- Creates a feature map highlighting where patterns appear

Example: Edge detection

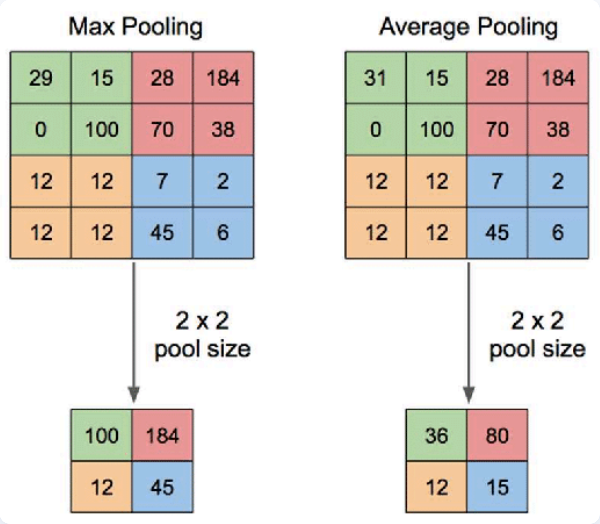
- Filter: $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$
- Detects vertical edges
- Different filters detect different patterns

Goal of learning (with backpropagation): Finding the best filters

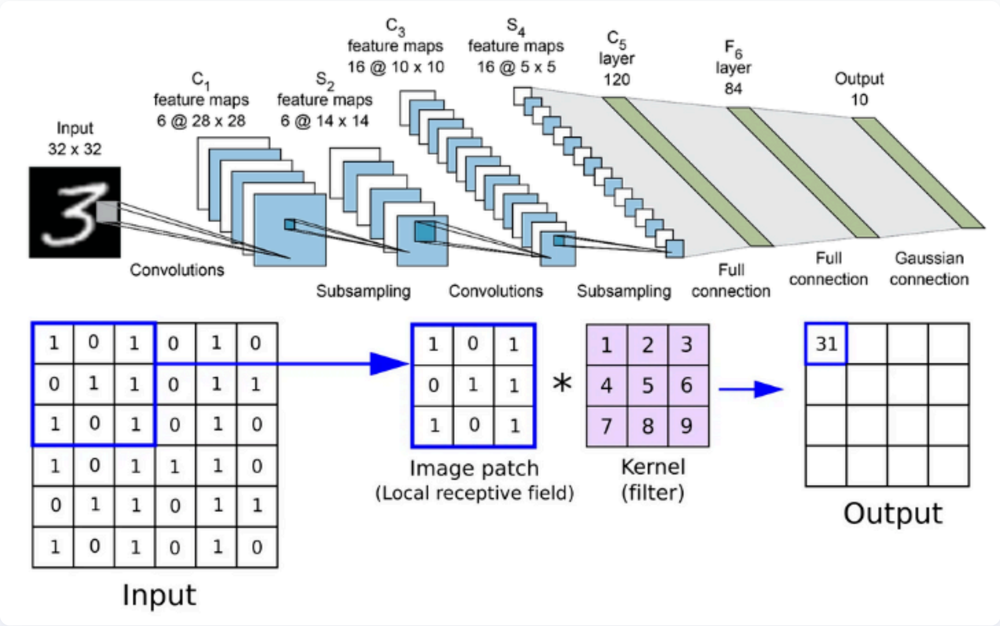
CNN Components: Pooling

Pooling Layer:

- Reduces the size of feature maps
- Provides some position invariance
- Reduces computation and prevents overfitting
- Common approach: Max pooling (take maximum value in region)



Complete CNN Architecture



Typical CNN architecture:

1. Input layer (image)
2. Convolutional layer + activation
3. Pooling layer
4. Repeat 2-3 several times
5. Flatten layer
6. Fully connected layers
7. Output layer

Each convolutional layer:

- Uses multiple filters to detect different patterns
- Creates multiple feature maps

CNN Example: Image Classification

LeNet-5 (one of the first CNNs):

- Developed by Yann LeCun in 1998
- Used for digit recognition (MNIST dataset)
- Structure:
  - Input: 32×32 grayscale image
  - Conv1: 6 filters of size 5×5
  - Pool1: 2×2 max pooling
  - Conv2: 16 filters of size 5×5
  - Pool2: 2×2 max pooling
  - FC1: 120 neurons
  - FC2: 84 neurons
  - Output: 10 neurons (one per digit)

Modern CNN Architectures

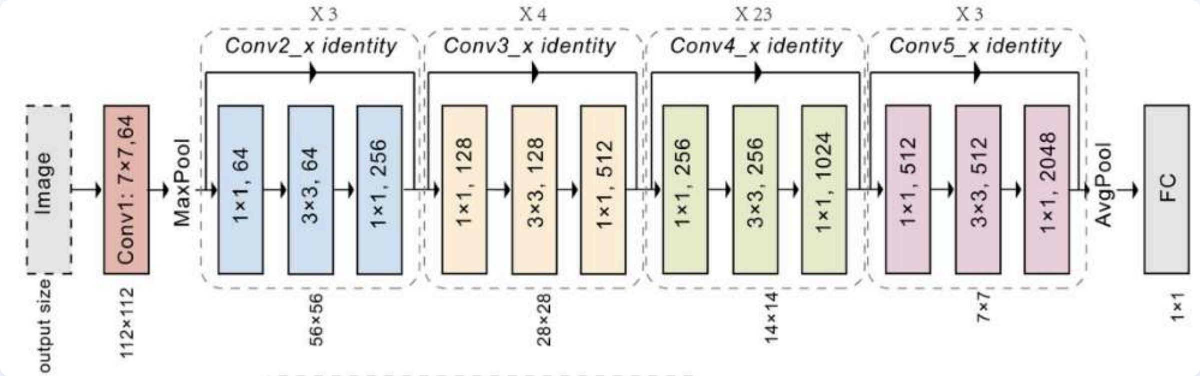
Deep CNNs that revolutionized computer vision:

AlexNet (2012):

- Won ImageNet competition, sparked deep learning revolution
- 8 layers, 60 million parameters
- Used ReLU activation, dropout regularization

ResNet (2015):

- Introduced residual connections to solve vanishing gradient
- Enabled much deeper networks (up to 152 layers)
- Structure: $F(x) + x$ instead of just $F(x)$



Jupyter notebook time!

Go to the course homepage: https://la7.lu/sk25

Click on "Colab 2"

Transfer Learning with CNNs

Concept:

- Take a pre-trained CNN (on massive dataset like ImageNet)
- Remove the final classification layer
- Add new layers for your specific task
- Fine-tune on your smaller dataset

Benefits:

- Requires much less data
- Much faster training
- Better performance

Common approach:

- Freeze early layers (generic features)
- Only train later layers (task-specific features)

Practical Model Development Process

Iterative process:

1. Start simple (baseline model)
2. Analyze errors
3. Incrementally add complexity
4. Monitor validation performance
5. Apply regularization as needed
6. Repeat until satisfactory performance

Hyperparameter tuning:

- Learning rate
- Network architecture (layers, neurons)
- Regularization strength
- Optimization algorithm

Debugging Neural Networks

Common issues and solutions:

Model doesn't learn (flat loss):

- Check for errors in data preprocessing
- Verify loss function implementation
- Try a larger learning rate

Model learns then plateaus:

- Try a different architecture
- Add regularization
- Implement learning rate schedules

Model is unstable (loss jumps around):

- Decrease learning rate
- Try gradient clipping
- Use a different optimizer (Adam)

Key Takeaways

1. Deep networks learn hierarchical representations of data
2. Convolutional Neural Networks excel at image-related tasks
3. Modern architectures solve training challenges in deep networks
4. Transfer learning enables effective use with limited data

Quiz time

Go to the course homepage: https://la7.lu/sk25

Click on "Quiz 5"