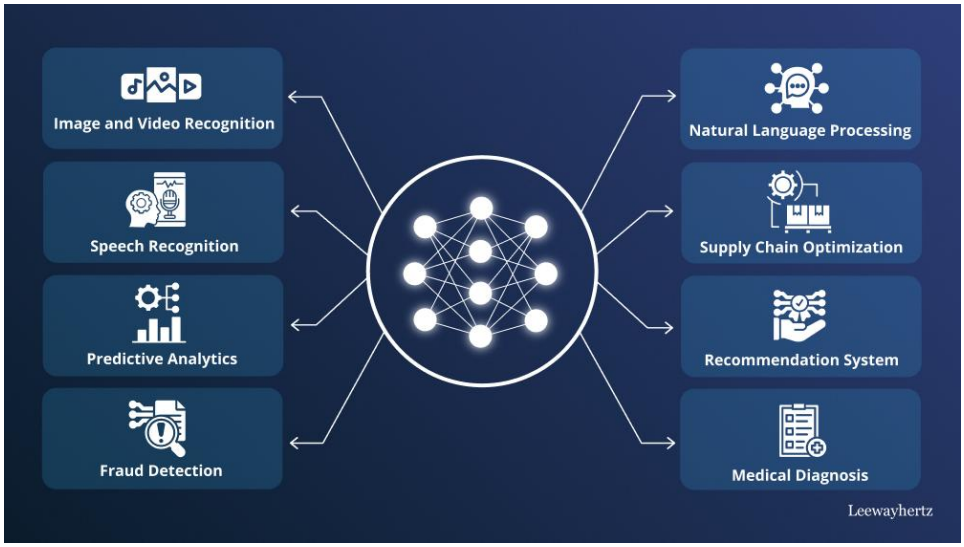
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Larbi Ben M'hidi University - Oum El Bouaghi
Faculty of Exact Sciences and Natural and Life Sciences
Department of Mathematics and Computer Science

DEEP LEARNING

COURSE HANDBOOK



Master's Degree - First Year

Specialty: Artificial Intelligence

Academic Year 2025-2026

Dr. ZERTAL Soumia

Preamble

This Deep Learning course handbook represents a comprehensive and rigorous exploration of one of artificial intelligence's most transformative disciplines, specifically designed for first-year Master's students specializing in Artificial Intelligence. The curriculum provides systematic coverage of both foundational principles and cutting-edge architectures that define contemporary deep learning practice.

The course adopts a carefully structured pedagogical progression, beginning with theoretical foundations including neural network mathematics, activation functions, and backpropagation mechanisms, before advancing through specialized architectures: convolutional neural networks for visual data processing, recurrent networks (RNNs, LSTMs, GRUs) for sequential and temporal analysis, and generative models (GANs and VAEs) for data synthesis and augmentation. Each chapter integrates mathematical rigor with practical implementation, featuring real-world applications spanning computer vision, natural language processing, biomedical signal analysis, and autonomous systems.

Students will develop essential competencies in designing, implementing, training, and critically evaluating deep neural networks while navigating architectural tradeoffs for domain-specific applications. The course emphasizes hands-on learning through exercises, projects, and implementation using modern frameworks, fostering both technical proficiency and conceptual intuition.

Prerequisites include solid foundations in linear algebra, calculus, probability theory, and programming (preferably Python), along with basic machine learning concepts. Beyond technical mastery, the course instills critical awareness of ethical dimensions; fairness, transparency, privacy, societal impact, and environmental costs; that accompany powerful AI technologies. As future practitioners shaping artificial intelligence's trajectory, students are encouraged to approach these transformative capabilities with both excellence and responsibility, cultivating habits of continuous learning, rigorous thinking, and ethical innovation in a rapidly evolving field.

Course Information

- **Master's Program Title:** Artificial Intelligence and Data Science
- **Semester:** S2
- **Teaching Unit:** UEF1
- **Course Title:** Deep Learning
- **Credits:** 06
- **Coefficient:** 03

1. **Learning Objectives**

The objective of the Deep Learning course is to provide students with a thorough understanding of the architectures, algorithms, and fundamental techniques of deep learning, enabling them to effectively leverage deep neural networks to solve complex problems in artificial intelligence and data science.

2. **Recommended Prerequisites:** Machine Learning

3. Course Content

Chapter 1: Fundamental Concepts

- Definition of deep learning
- Neuron architecture
- Neural network structures (layers, connections)
- Gradient backpropagation mechanisms

Chapter 2: Convolutional Neural Networks (CNN)

- Convolutions and pooling
- Typical CNN architecture
- Applications in computer vision

Chapter 3: Recursive and Recurrent Neural Networks (RNN)

- Sequence concepts
- RNN architecture
- Applications in natural language processing (NLP)

Chapter 4: Generative Adversarial Networks (GAN)

- Introduction to GANs
- Generative training
- Applications in image and content generation

Chapter 5: Autoencoders and Variational Autoencoders (VAE)

- Autoencoder principles
- VAE for data generation
- Applications in data compression and generation

Table of Contents

Chapter 01

Deep Learning – Theoretical Foundations, Architectures and Advanced Applications

1. General Introduction

Deep learning is now one of the most influential disciplines in artificial intelligence (AI). It represents a major evolution of classical artificial neural networks, made possible by the exponential increase in computing power, the availability of massive datasets (Big Data), and recent algorithmic advances.

Unlike traditional machine learning approaches, where feature extraction is often performed manually, deep learning enables the automatic learning of hierarchical representations directly from raw data. This capability gives deep models remarkable efficiency in complex tasks such as computer vision, natural language processing, and autonomous decision-making.

2. History and evolution of deep learning

The origins of neural networks date back to the 1950s with Rosenblatt's perceptron. However, theoretical (such as the XOR problem) and material limitations have led to a slowdown in research.

The resurgence of Deep Learning since the 2000s can be explained by:

1. The rise of graphics processing units (GPUs);
2. The availability of large-scale databases;
3. The introduction of new efficient architectures (CNN, RNN).

These advances have led to performances exceeding those of humans in certain specific tasks.

3. Definition of deep learning

3.1. Formal Definition

Deep learning is a subfield of machine learning based on the use of deep artificial neural networks, composed of several hidden layers. Mathematically, these models seek to approximate a complex function $f(X) = Y$ relating an input space (X) to an output space (Y), from a training dataset.

Each layer of the network learns a non-linear transformation of the data, allowing the progressive construction of increasingly abstract representations.

3.2. Difference between machine learning and deep learning

In machine learning, model performance depends heavily on the quality of manually extracted features. In contrast, deep learning automates this step using multi-layered architectures capable of simultaneously learning features and the decision model.

Comparative Table: Machine Learning vs Deep Learning

Criterion	Classical Machine Learning	Deep Learning
Feature Extraction	Manual (feature engineering)	Automatic (learned by the network)
Required Data Volume	Can work with small datasets	Requires large amounts of data
Computational Power	CPU is sufficient in most cases	GPU/TPU is essential for large models
Interpretability	Often interpretable (e.g., decision trees)	Black-box models, difficult to interpret
Performance on Tabular Data	Good (e.g., XGBoost, Random Forest)	Not always superior
Performance on Images/Text/Audio	Limited without manual feature engineering	Excellent, state-of-the-art performance
Training Time	Fast (minutes to hours)	Long (hours to days)
Examples of Algorithms	SVM, Random Forest, k-NN	CNN, RNN, Transformer, GAN, VAE

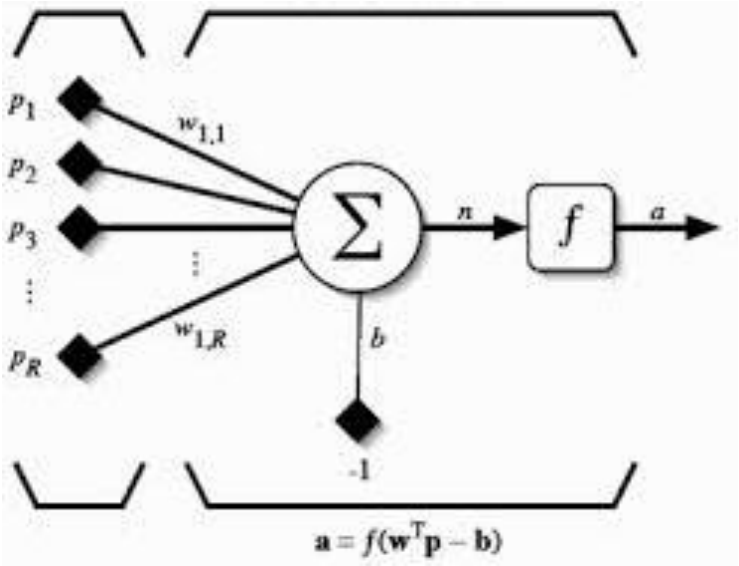
4. Functioning of deep neural networks

4.1. Mathematical model of an artificial neuron

An artificial neuron performs a linear combination of its inputs followed by a non-linear activation function:

$$z = \sum_{i=1}^{n} wi * xi + b$$

$$y = f(z)$$



Where :

xi are the inputs, wi the synaptic weights, b the bias and f(x) is the activation function.

4.2. Activation function

Activation functions introduce the non-linearity necessary for learning complex relationships. The most commonly used are:

- Sigmoid
- Hyperbolic tangent
- ReLU (Rectified Linear Unit)
- Softmax (for multi-class classification)

Comparative Table of Activation Functions

Function	Formula	Range	Advantages	Disadvantages	Typical Use
Sigmoid	$\sigma(x) = 1 / (1 + e^{-x})$	[0, 1]	Probabilistic output	Vanishing gradient, not zero-centered	Binary output layer
Tanh	$\tanh(x) = (e^{x} - e^{-x}) / (e^{x} + e^{-x})$	[-1, 1]	Zero-centered	Vanishing gradient	RNNs, LSTMs
ReLU	$\max(0, x)$	$[0, +\infty)$	Simple, fast, no positive-side saturation	Dying ReLU (dead neurons)	CNNs, hidden layers
Leaky ReLU	$\max(0.01x, x)$	$(-\infty, +\infty)$	Prevents the Dying ReLU problem	Requires choosing the α slope	ReLU variant
ELU	x if $x > 0$, $\alpha(e^{x} - 1)$ otherwise	$(-\infty, +\infty)$	Allows negative outputs	Exponential computation cost	Deep neural networks
GELU	$x \cdot \Phi(x)$	$(-\infty, +\infty)$	High performance, smooth activation	Computationally complex	Transformers, BERT, GPT
Swish	$x \cdot \sigma(x)$	$(-\infty, +\infty)$	Self-gated, smooth	Higher computational cost	EfficientNet
Softmax	$e^{xi} / \Sigma e^{xj}$	[0, 1] (sum = 1)	Produces a probability distribution	Can saturate with very large values	Multi-class output layer

4.3. Learning and backpropagation of errors

The learning process relies on minimizing a loss function, which measures the difference between the predicted and actual output. The backpropagation algorithm calculates the gradient of this function with respect to the network parameters, allowing them to be updated via optimization methods such as gradient descent.

4.4. Optimizers

The choice of optimizer has a significant impact on both the convergence speed and the final performance of a model. Below are the main optimizers used in deep learning.

Stochastic Gradient Descent (SGD)

SGD is the most fundamental optimization algorithm. The weights are updated according to:

$$\theta \leftarrow \theta - \eta \cdot \nabla L(\theta)$$

where η is the learning rate.

The momentum variant accelerates convergence:

$$v \leftarrow \beta v - \eta \nabla L(\theta) v$$

Adam (Adaptive Moment Estimation)

Adam combines the advantages of RMSprop and momentum. It is the default optimizer in the majority of deep learning applications.

$$m \leftarrow \beta_1 m + (1-\beta_1)\nabla L$$

$$v \leftarrow \beta_2 v + (1-\beta_2)(\nabla L)^2$$

$$\theta \leftarrow \theta - \eta \cdot \frac{m}{\sqrt{v} + \varepsilon}$$

with $\beta 1=0.9$, $\beta 2=0.999$, and $\varepsilon=10^{-8}$

Optimizer	Advantages	Disadvantages	Key Hyperparameters
SGD	Simple, strong inductive bias, often better generalization	Slow convergence, sensitive to learning rate	η (learning rate), momentum β
Adam	Fast convergence, adaptive, robust	May overfit, final learning rate may be suboptimal	η, β_1, β_2, ε
RMSprop	Well-suited for RNNs, adaptive	No momentum by default	η, ρ (decay rate)
AdaGrad	Effective for sparse data (NLP)	Learning rate decreases too quickly	Initial η
AdamW	Adam with corrected weight decay	Slightly higher computational cost	η, β_1, β_2, λ (weight decay)

4.5 Regularization Techniques

Regularization aims to reduce overfitting by constraining the complexity of a model.

L1 and L2 Regularization

L1 Regularization (Lasso) adds |w| to the loss function, promoting sparse weights.

$$L_{total}=L_{data}+ \lambda \sum|wi|$$

L2 Regularization (Ridge) adds w^2 to the loss function, encouraging small but nonzero weights.

$$L_{total}=L_{data}+ \lambda \sum wi^2$$

Dropout

During training, each neuron is randomly deactivated with probability ppp (typically between 0.2 and 0.5). This technique simulates an ensemble of subnetworks and forces the model to learn redundant and robust representations.

Batch Normalization

Batch Normalization normalizes the activations of a layer within a mini-batch. It accelerates convergence, reduces sensitivity to initialization, and acts as a regularizer.

Early Stopping

Early Stopping halts training as soon as validation performance stops improving. It prevents overfitting without modifying the model architecture.

5. Learning Paradigms in Deep Learning

5.1. **Supervised learning**

The model is trained on labeled data. This paradigm is dominant in classification and regression.

5.2. Unsupervised learning

It aims to discover the intrinsic structure of the data, notably through autoencoders and dimensionality reduction methods.

5.3. Reinforcement learning

An agent learns an optimal policy by interacting with a dynamic environment, maximizing a cumulative reward function.

6. Major architectures of deep learning

6.1. Convolutional Neural Networks (CNNs)

CNNs exploit convolution operations to extract local features and are particularly well-suited to spatially structured data, such as images.

6.2. Recurrent Neural Networks (RNNs)

RNNs are designed for processing sequential and time-domain data, such as signals or texts.

6.3. Advanced Architectures

Modern architectures include LSTM, GRU, and Transformers, which allow for better management of long-term dependencies.

7. Scientific and industrial applications

Deep learning has enabled major advances in numerous scientific and industrial fields. This section presents concrete examples and case studies to illustrate the real contribution of deep learning.

7.1. Vision: medical image classification

In the medical field, convolutional neural networks (CNNs) are widely used for analyzing radiological images (MRI, CT scans, X-rays). For example, a CNN can be trained to automatically detect pulmonary abnormalities from chest X-rays.

The process includes:

- The collection and annotation of medical images;
- Standardization and data augmentation;
- Training a deep CNN model;
- Evaluation using metrics such as accuracy, recall, and AUC.

These systems help doctors with early diagnosis and reduce human error.

7.2.Natural language processing: sentiment analysis

In natural language processing (NLP), deep models are used for sentiment analysis in social networks or customer reviews. Architectures such as RNNs, LSTMs, or Transformers learn to represent the meaning of a text by taking context into account.

For example, a Transformer-based model can automatically classify comments into positive, negative, or neutral sentiments, which is essential for strategic intelligence and opinion analysis.

7.3.Signal processing: automatic recognition of biomedical signals

In the analysis of biomedical signals (ECG, EEG), deep learning enables the automatic detection of anomalies and pathologies. CNNs and RNNs are combined to exploit both the morphological and temporal characteristics of the signals.

7.4.Standalone navigation

In robotics and autonomous vehicles, deep reinforcement learning allows an agent to learn how to navigate a complex environment. The agent receives rewards or penalties based on its actions and progressively improves its decision-making strategy. These methods are applied to trajectory planning, obstacle avoidance, and the optimization of autonomous behaviors.

8. Scientific challenges and current limitations

Key challenges include :

- Dependence on large amounts of data;
- The high computational cost;
- The lack of interpretability of the models;
- Algorithmic biases and ethical issues.
-

9. Research trends and perspectives

Current research is focused on federated learning, transfer learning, explainable models (XAI) and reducing the energy footprint of deep networks.

10. Conclusion

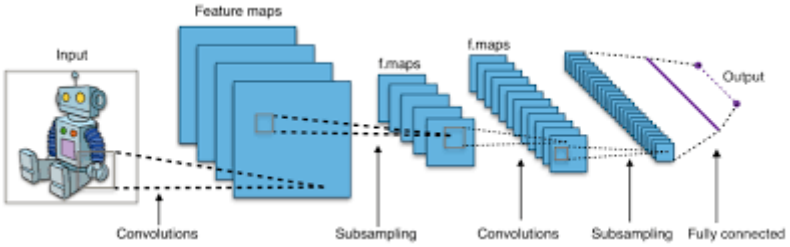
Deep learning represents a major scientific advancement in artificial intelligence. A thorough understanding of it is essential for designing high-performing, robust, and ethically responsible intelligent systems. The examples and case studies presented in this chapter

demonstrate the concrete impact of deep learning in diverse fields, ranging from healthcare to autonomous systems.

Future developments will continue to expand its scope while posing new research challenges, particularly in terms of interpretability, energy efficiency, and reliability.

Chapter 02

Convolutional Neural Networks: Foundations and Applications



1. Introduction

The emergence of Convolutional Neural Networks (CNNs) marks a watershed moment in the history of artificial intelligence and computer vision. Since their introduction by Yann LeCun and colleagues in the late 1980s, and their spectacular resurgence following the ImageNet breakthrough in 2012, CNNs have fundamentally transformed our approach to visual data processing and understanding. Today, these architectures power critical applications ranging from medical diagnosis and autonomous vehicles to facial recognition and content moderation systems.

2. The Challenge of Visual Data Processing

Visual data presents unique challenges that distinguish it from other data modalities. Images are inherently high-dimensional: a modest 224×224 pixel color image contains over 150,000 values. This dimensionality poses severe computational and statistical challenges for traditional fully-connected neural networks. Moreover, visual information exhibits distinct structural properties that standard architectures fail to exploit effectively.

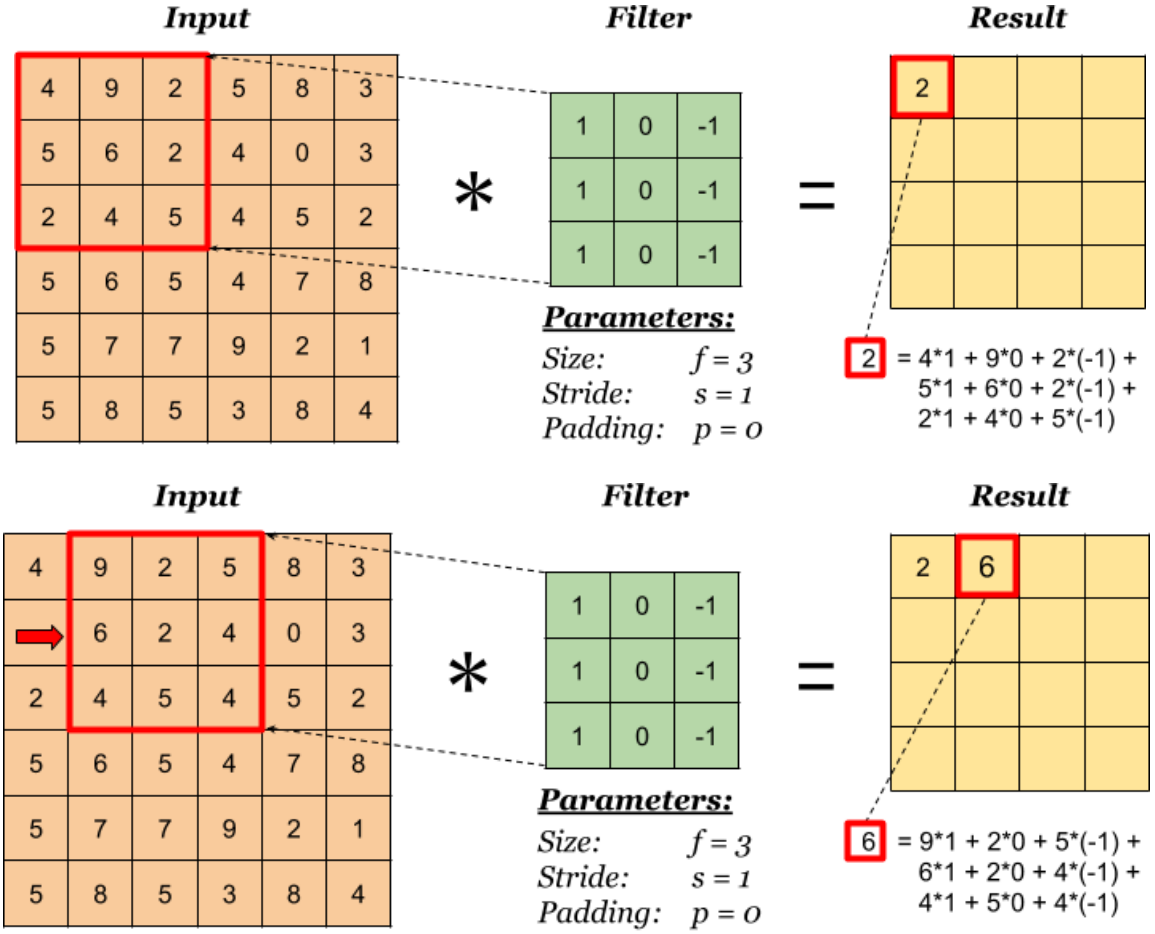
Consider a simple example: the task of recognizing a cat in an image. A fully-connected network treating each pixel as an independent feature would require millions of parameters, making it computationally prohibitive and highly susceptible to overfitting. More critically, such an architecture would fail to leverage three fundamental properties inherent to visual data:

- **Spatial locality**: Pixels that are close together are more likely to be related than distant pixels. The whiskers, eyes, and nose of a cat form local patterns that are meaningful for recognition.
- **Translation invariance**: A cat remains a cat whether it appears in the top-left corner or the bottom-right corner of an image. The features that identify a cat should be detectable regardless of position.
- **Hierarchical composition**: Visual understanding naturally proceeds from simple features (edges, textures) to intermediate structures (eyes, ears) to complex objects (faces, animals). Effective visual processing should mirror this hierarchical organization.

3. Convolutional Neural Networks (CNN) ou ConNet

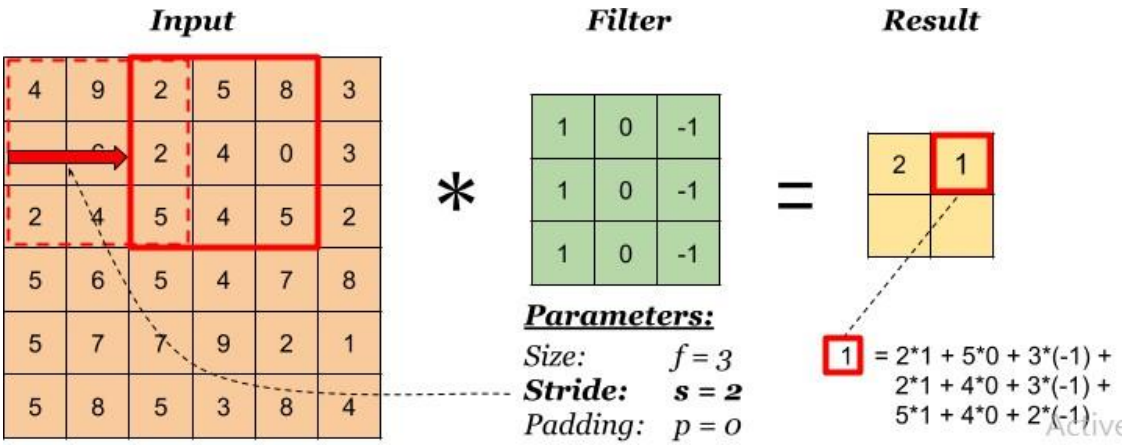
The term 'convolutional' derives from the matrix convolution operation used in signal processing. In ConvNets, two new types of layers have been introduced into the network architecture: the convolutional layer and the pooling layer

Convolution Operation



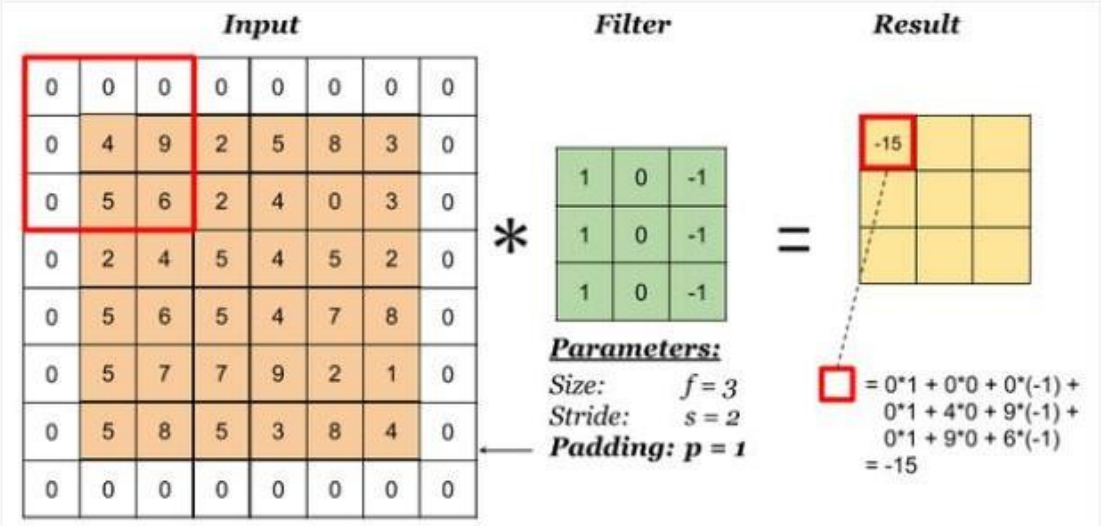
The total number of multiplications to calculate the result above is (4 x 4) x (3 x 3) = 144.

Stride

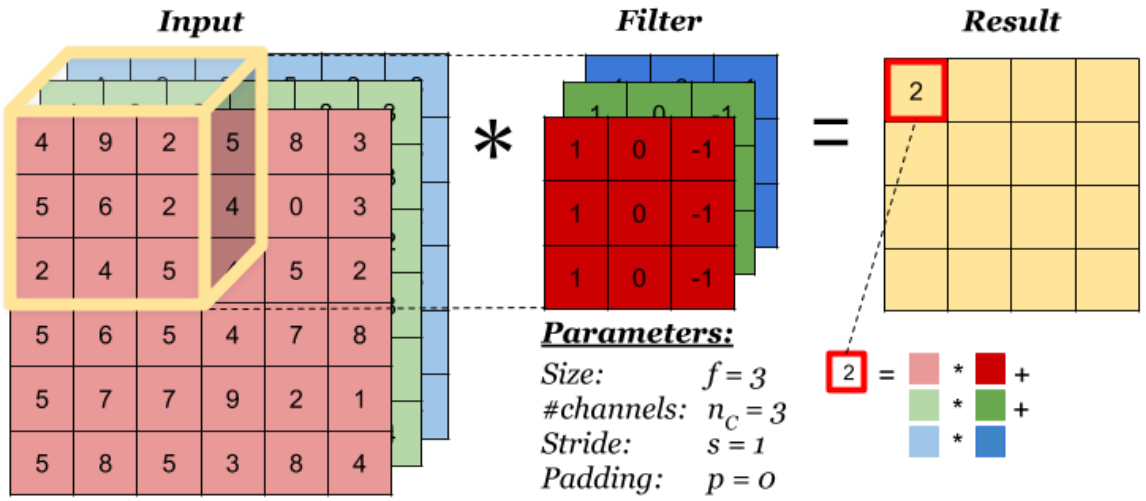


The total number of multiplications to calculate the result above is (2 x 2) x (3 x 3) = 36.

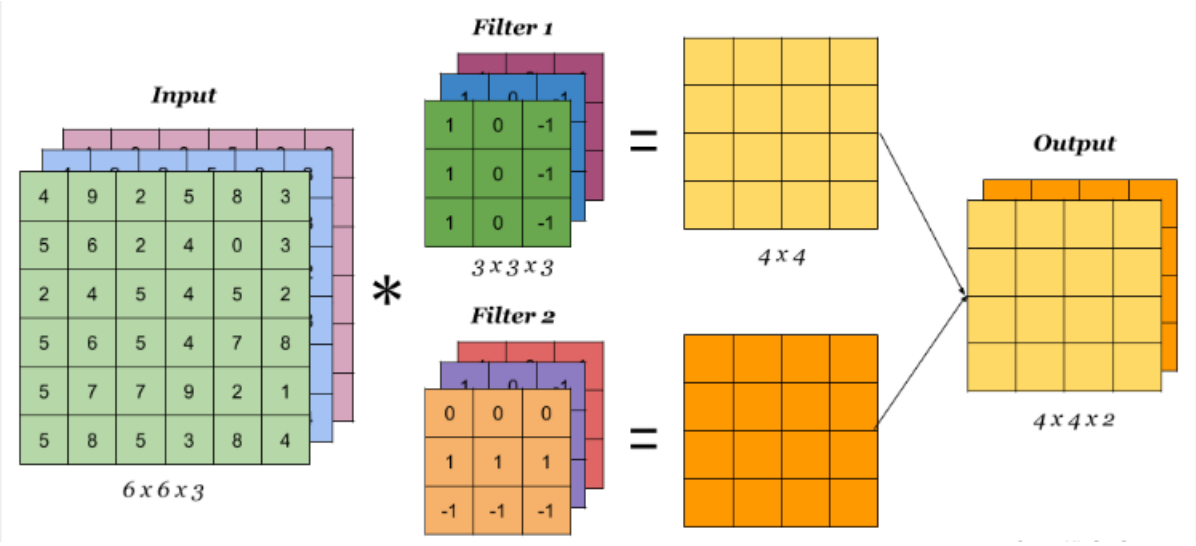
Padding



4. Convolution Operation on Volume



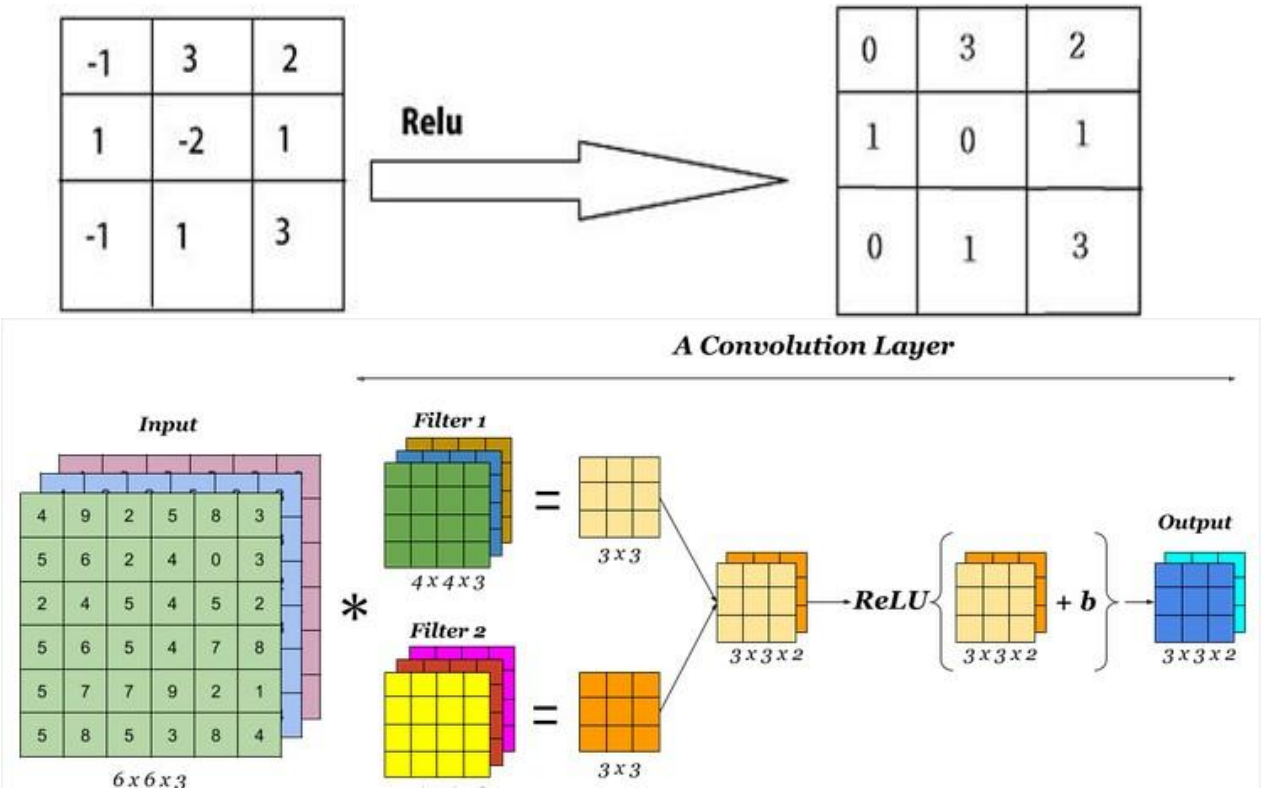
5. Convolution Operation with Multiple Filters



The total number of multiplications to calculate the result is (4 x 4 x 2) x (3 x 3 x 3) = 864.

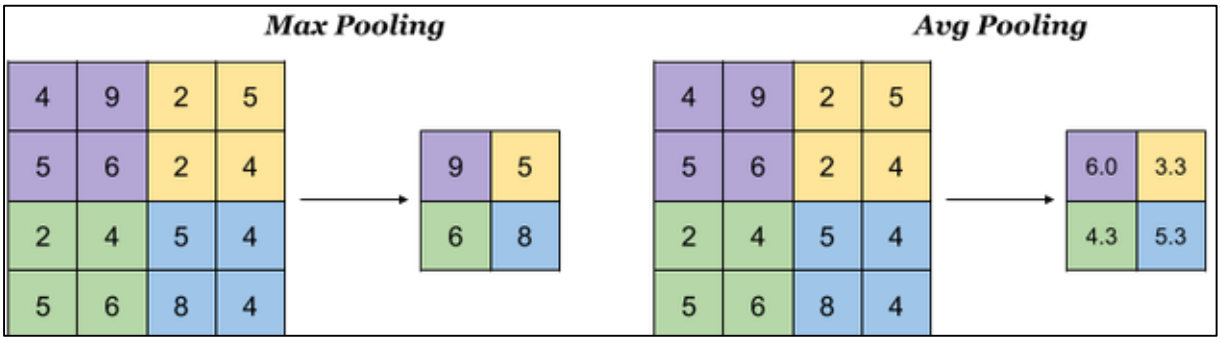
6. One Convolution Layer

Finally to make up a convolution layer, a bias (ϵ R) is added and an activation function such as **ReLU** or **tanh** is applied.



Pooling Layer

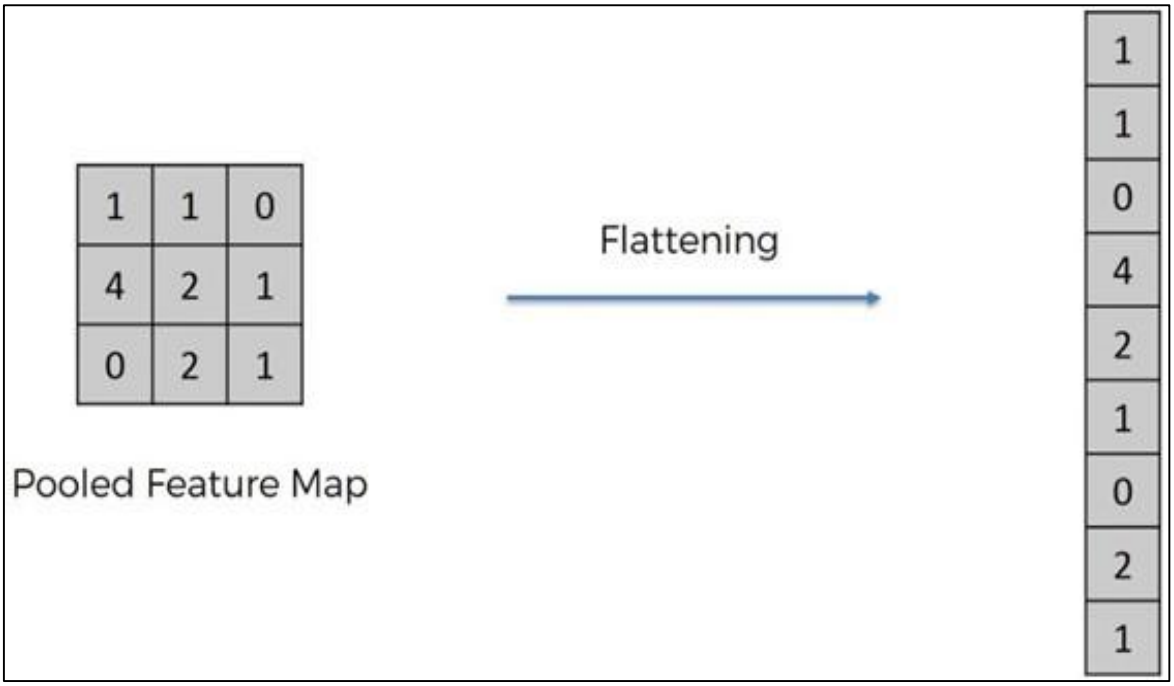
Pooling layer is used to reduce the size of the representations and to speed up calculations. Sample types of pooling are max pooling and avg pooling.



Flattening

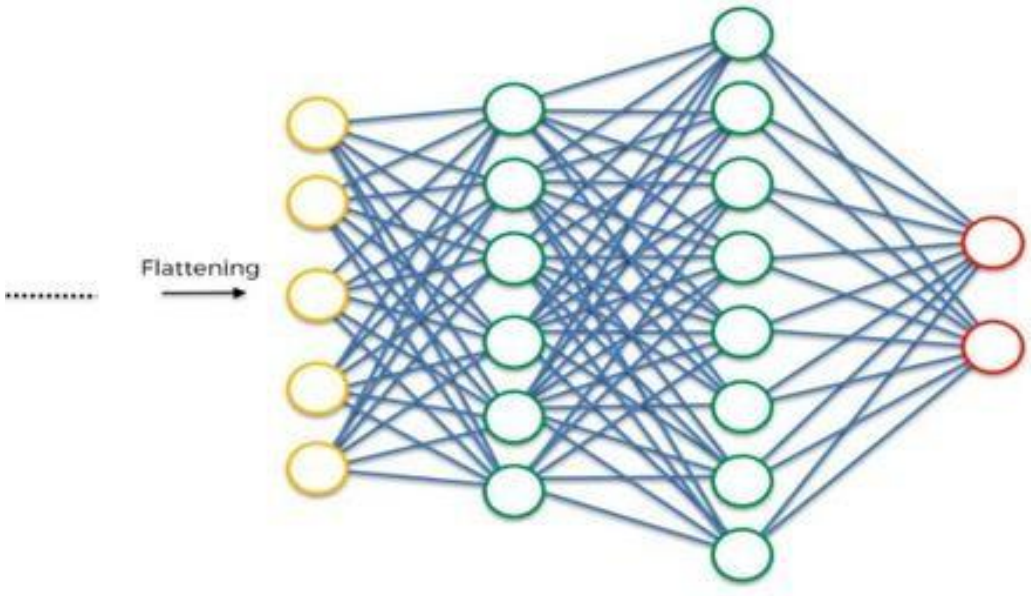
Flattening is the process of converting al the resultant 2-dimensional arrays into a single long continuous linear vector.

So Flattening is become the input of Artificial Neural Network.



7. Fully Connected Layer (FC Layer) or classification

The Fully Connected layer is a traditional multi-Layer Perceptron that uses a softmax activation function in the output layer. The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

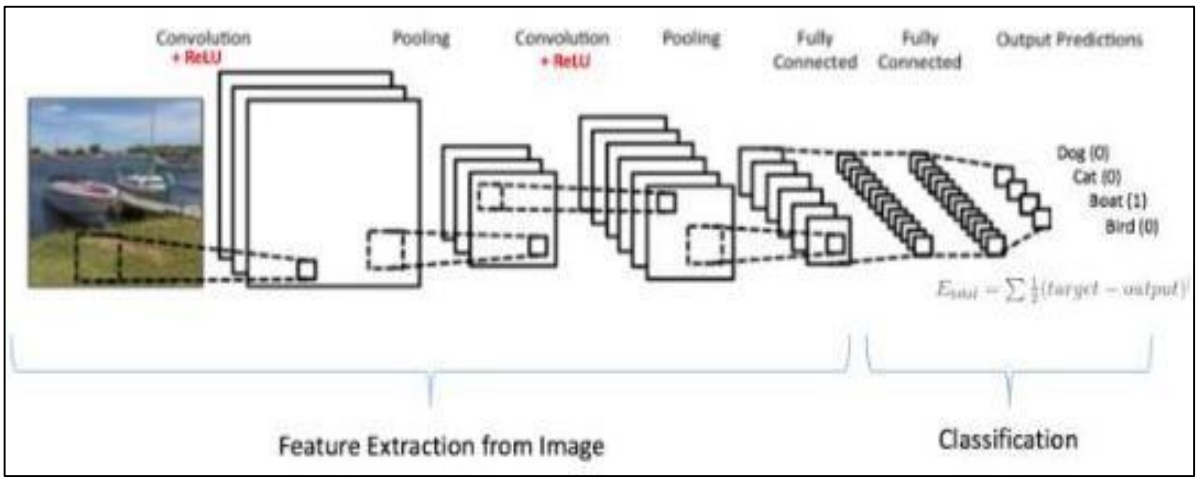


The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset

8. Training process of the CNN

The convolution + Pooling layers act as Features extractor from the input image while a fully connected layer acts as the classifier.

Input Image = Boat ; Target Vector = [0, 0, 1, 0]



Step1

Initialization all filters and parameters/weights with random values.

Step2

The network takes a training image as input, goes through the forward propagation step (convolution, ReLU, and pooling operations along with forwarding propagation in the Fully Connected layer) and finds the output probabilities for each class. For example, let be the output probabilities for the boat image are [0.2, 0.4, 0.1, 0.3]

Step3

Calculate the total error at the output layer (summation over all 4 classes)

$$\text{Total Error} = \sum \tfrac{1}{2} (\text{target probability} - \text{output probability})^2$$

Step4

Use Backpropagation to calculate the gradients of the error concerning all weights in the network and use gradient descent to update all filter values/weights and parameter values to minimize the
output error.

When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0]. This means that the network has learned to classify this particular image correctly by adjusting its weights/filters such that the output error is reduced.

Parameters : number of filters, filter sizes, architecture of the network have all been fixed before Step 1 and do not change during the training process — only the values of the filter matrix and connection weights get updated.

Step5

Repeat steps 2–4 with all images in the training set.

9. Transfer Learning

Transfer learning embodies the principle of leveraging knowledge acquired by a neural network during the resolution of one problem to solve another, potentially related problem. This knowledge transfer, which gives rise to the term "transfer learning," enables models to benefit from representations learned on large-scale datasets and apply them to new domains or tasks. Two primary transfer learning strategies are commonly employed:

9.1. ConvNet Feature Extraction

In this strategy, the ConvNet functions as a fixed feature extractor. A feature vector is extracted from a designated layer of the pre-trained model without any modification to the network's architecture or learned weights. The extracted feature representation is subsequently utilized as input for training a new classifier or model for the target task. This method is particularly effective when the new task has limited training data or when the pre-trained features are sufficiently general.

9.2. ConvNet Fine-Tuning

Fine-tuning involves initializing a new ConvNet with both the weights and architectural structure of a pre-trained model. The pre-trained architecture is adapted for the new task—typically through modification of the final layers to match the target task's output requirements. The adapted model is then trained on the new task data, with gradient updates applied either to the entire network or selectively to specific layers. This approach allows the model to adapt learned representations to the specificities of the new task while retaining beneficial knowledge from pre-training.

9.3. Comparative Table of Pretrained CNN Architectures

Architecture	Year	Parameters	ImageNet Top-1 Accuracy	Key Features
LeNet-5	1998	~60K	—	First modern CNN, designed for MNIST digit recognition
AlexNet	2012	~60M	63.3%	Introduced ReLU, Dropout, GPU training; marked the beginning of the deep learning era
VGG-16	2014	~138M	74.4%	Stacked 3×3 filters; simple and effective architecture
GoogLeNet	2014	~6.8M	74.8%	Inception modules; achieves high performance with very few parameters
ResNet-50	2015	~25M	76.1%	Residual (skip) connections enabling very deep networks
DenseNet-121	2016	~8M	74.9%	Dense connections between all layers
MobileNetV2	2018	~3.4M	72.0%	Lightweight architecture optimized for mobile and embedded devices
EfficientNet-B0	2019	~5.3M	77.1%	Compound scaling of depth, width, and resolution

9.4. ResNet — Residual Connections (Skip Connections)

The degradation problem: in very deep networks, performance degrades even without overfitting. ResNet solves this issue using residual connections.

Key idea: instead of learning the mapping H(x), the network learns the residual function

$$F(x)=H(x)-x$$

This identity shortcut connection allows gradients to flow through many layers without vanishing, making it possible to train networks with 50, 100, or even 152 layers.

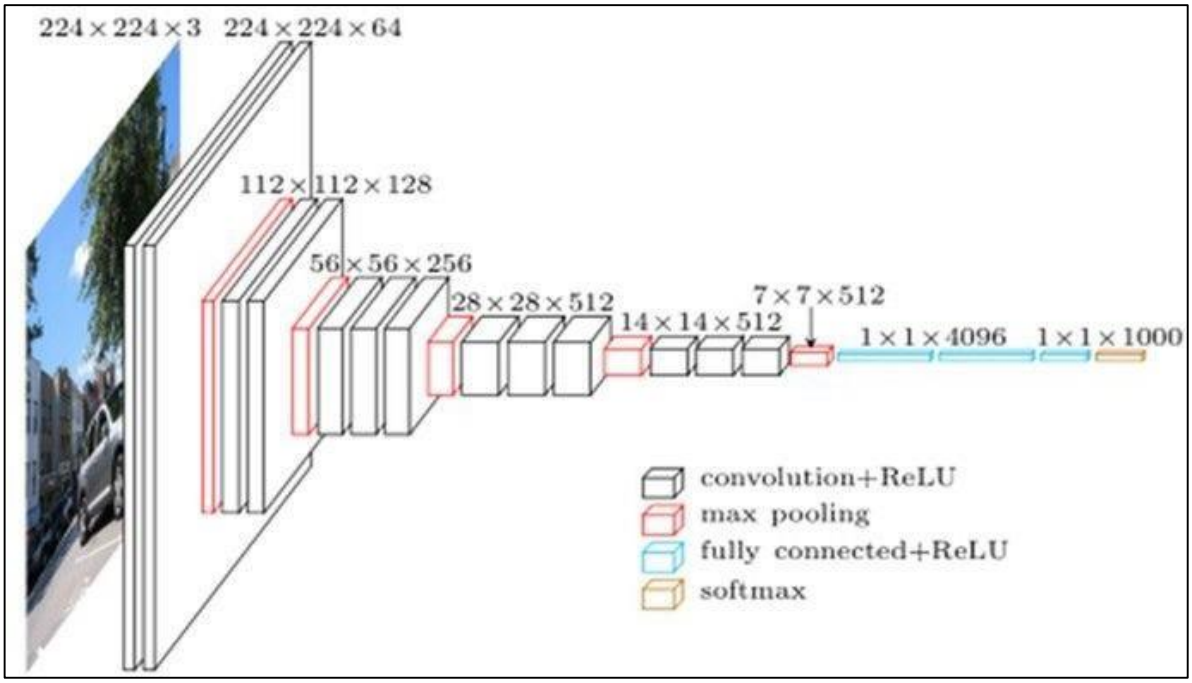
9.5. Transfer Learning Implementation in Keras

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models

# Load ResNet50 pre-trained on ImageNet
base_model = ResNet50(weights='imagenet', include_top=False,
                      input_shape=(224, 224, 3))

# Freeze base layers (feature extraction)
base_model.trainable = False

# Add new classification layers
model = models.Sequential([
   base_model,
   layers.GlobalAveragePooling2D(),
   layers.Dense(256, activation='relu'),
   layers.Dropout(0.5),
   layers.Dense(num_classes, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Fine-tuning: unfreeze last 20 layers
base_model.trainable = True
for layer in base_model.layers[:-20]:
   layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
```

```
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

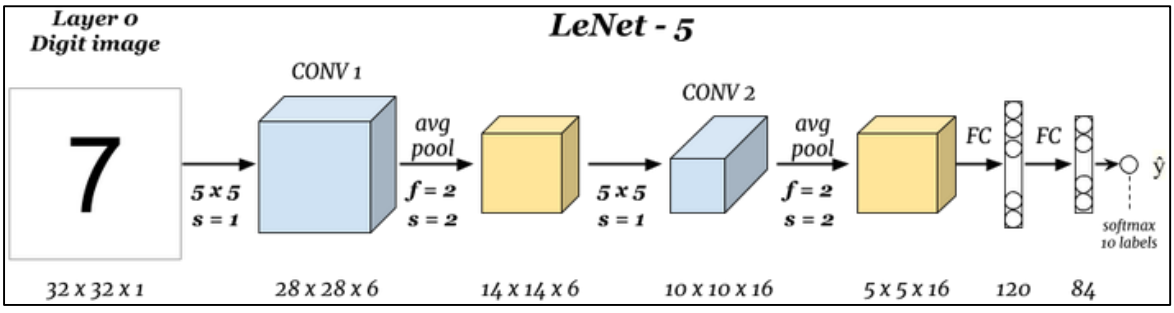
10. Examples

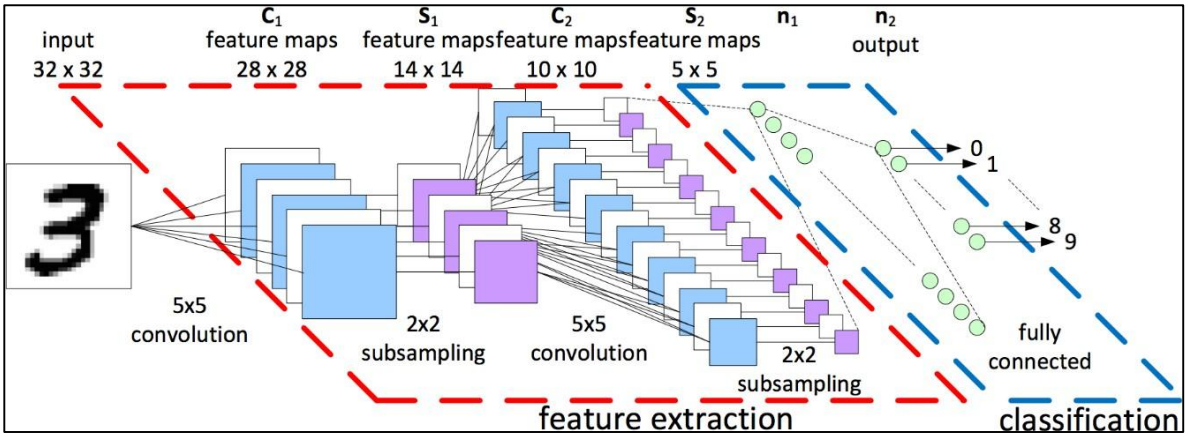
10.1.3D representation of the VGG-16 architecture



from keras.applications.vgg16 import VGG16

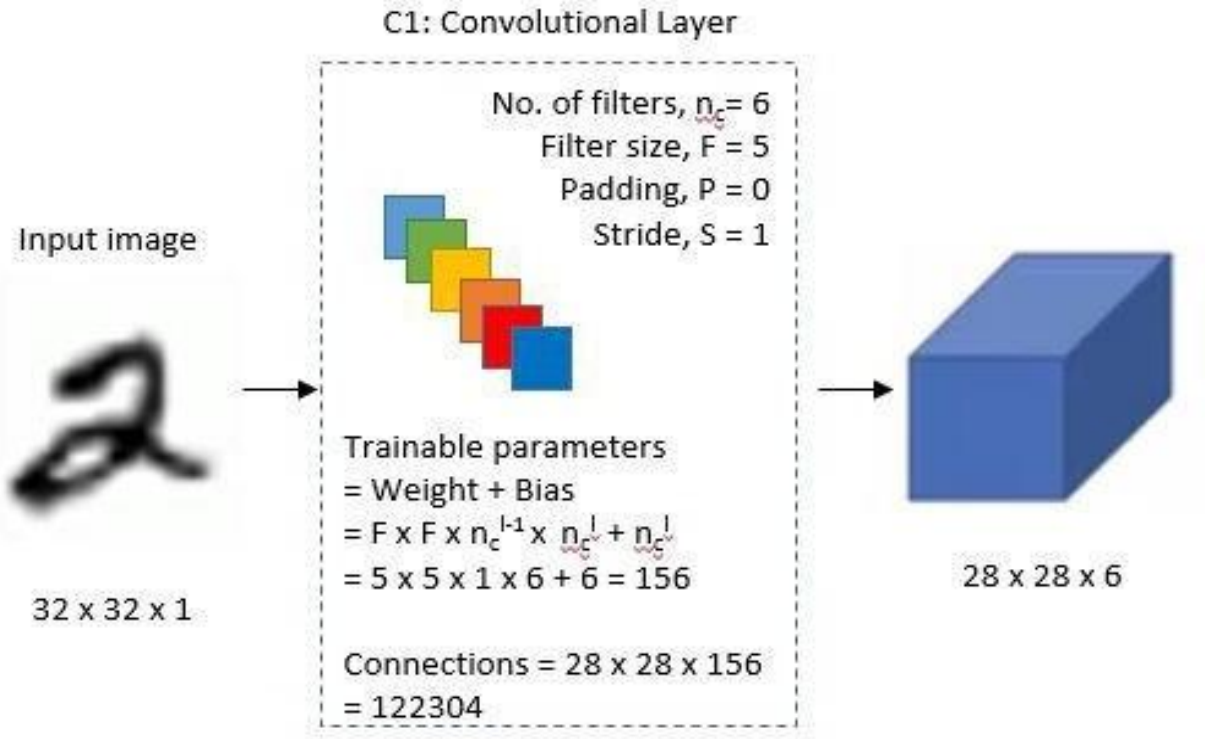
10.2.Classic Network: LeNet – 5





10.2.1. First Layer : Convolution layer C1

6 filtres (or feature maps) of size 5x5, Stride=1, Padding= 0



The input for LeNet-5 is a 32×32 grayscale image which passes through the first convolutional layer with 6 filters (or feature maps) having size 5×5 and a stride = 1.

The image dimensions changes from 32x32x1 to 28x28x6.

$$\text{OutputWidth} = \left(\frac{W - Fw + 2P}{Sw}\right) + 1$$

$$\textbf{OutputWidth} = \left(\frac{\mathbf{32 - 5 + 0}}{\mathbf{1}}\right) + \mathbf{1} = \mathbf{27 + 1 = 28}$$

$$\text{OutputHeight} = \left(\frac{H - Fh + 2P}{Sh}\right) + 1$$

$$\boldsymbol{OutputHeight} = \left(\frac{\mathbf{32-5+0}}{\mathbf{1}}\right) \mathbf{+1 = 27+1 = 28}$$

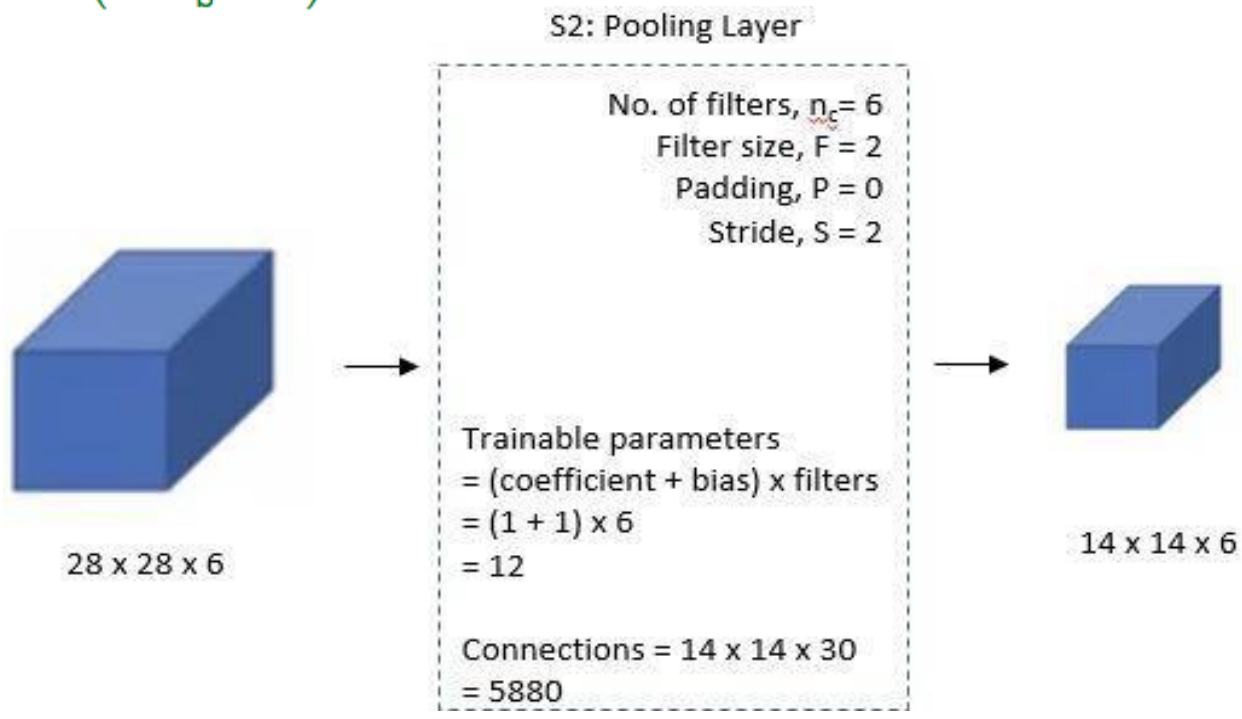
```
trainable parameters = (weight * input maps + bias) * feature maps
                     = (5 * 5 * 1 + 1) * 6 = 156
```

```
connections = (input + bias) * feature maps * feature map size
            = (5 * 5 + 1) * 6 * 28 * 28 = 122304
```

10.2.2. 2nd layer : Subsampling (Pooling) layer (S2)

6 Filtres (Feature maps) of size 2x2, Stride=2, Padding = 0

$$OM = \left(\frac{IM + 2P - F}{S}\right) + 1$$

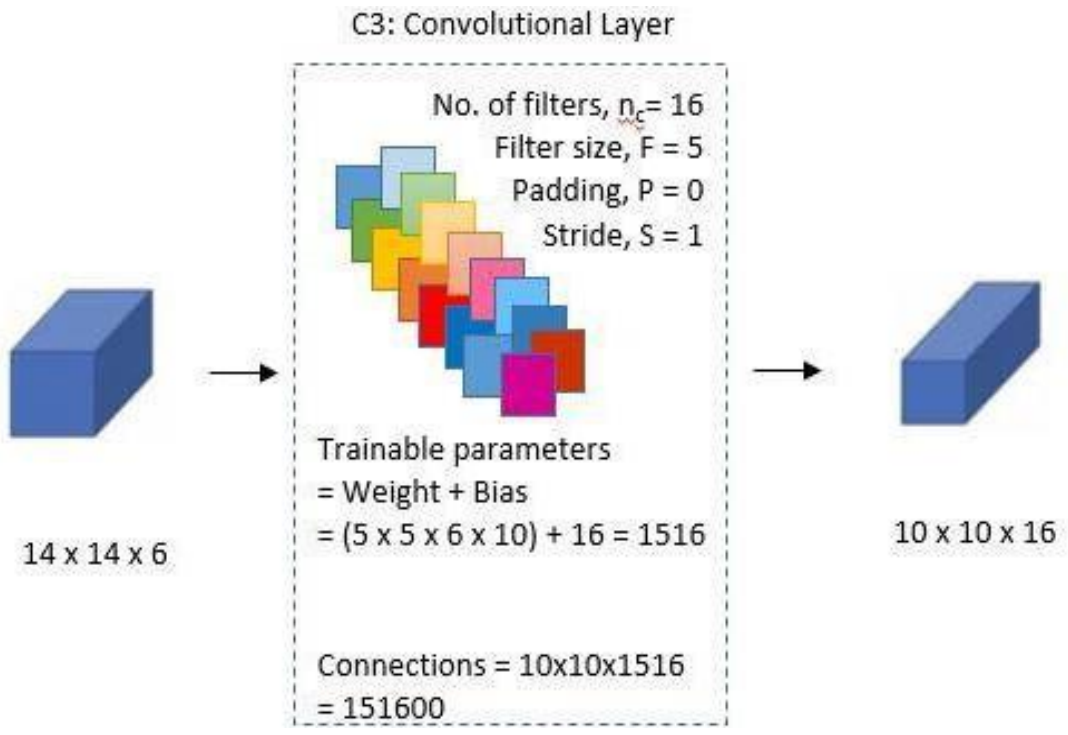


$$OM = \left(\frac{28 + 2 * 0 - 2}{2}\right) + 1 = 14; Output\ Matrix = 14 * 14$$

```
trainable params = (weight + bias) * feature maps
                 = (1 + 1) * 6 = 12
```

```
connections = (input + bias) * feature maps * feature map size
            = (2 * 2 + 1) * 6 * 14 * 14 = 5880
```

10.2.3. 3rd layer : Convolution layer (C3)



10.2.4. Calculate Feature Map

$$\text{OutputWidth} = \left(\frac{W - Fw + 2P}{Sw}\right) + 1$$

OutputWidth = (14 – 5 + 0 / 1) + 1 = 10

$$\text{OutputHeight} = \left(\frac{H - Fh + 2P}{Sh}\right) + 1$$

OutputHeight = (14 – 5 + 0 / 1) + 1 = 10. Output Matrix = 10 x 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

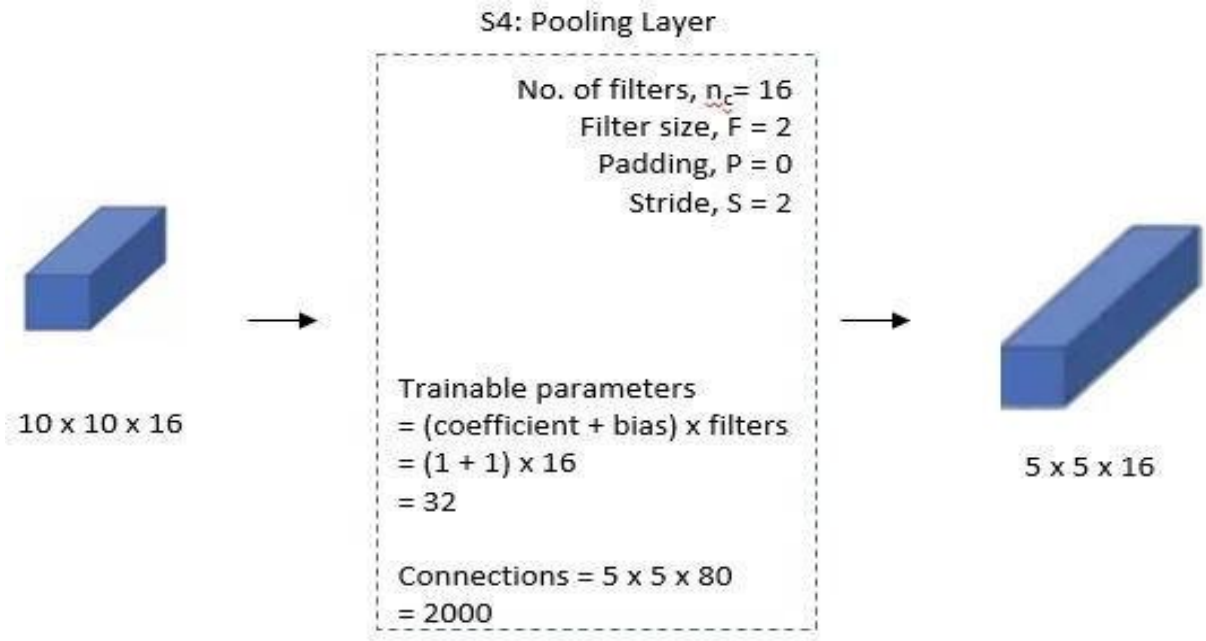
EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

- First 6 feature maps are connected to 3 contiguous input maps each (overlapping 2 maps)
- Second 6 feature maps are connected to 4 contiguous input maps (overlapping 3 maps)
- Next 3 feature maps are connected to 4 discontinuous input maps (overlapping 1 map)
- Last 1 feature map are connected to all 6 input maps

```
trainable params = (weight * input maps + bias) * feature maps

   1st group = (5 * 5 * 3 + 1) * 6 = 456
   2nd group = (5 * 5 * 4 + 1) * 6 = 606
   3rd group = (5 * 5 * 4 + 1) * 3 = 303
   4th group = (5 * 5 * 6 + 1) * 1 = 151

All group = 456 + 606 + 303 + 151 = 1516
```

```
connections = (input + bias) * feature maps * feature map size
            = trainable params * feature map size
            = 1516 * 10 * 10 = 151600
```

10.2.5. 4th Layer : Pooling Layer S4



$$OM = (\frac{IM+2P-F}{S}) +1$$

$$OM = \left(\frac{10 + 2 * 0 - 2}{2}\right) + 1 = 5\,; Output\ Matrix = 5x5$$

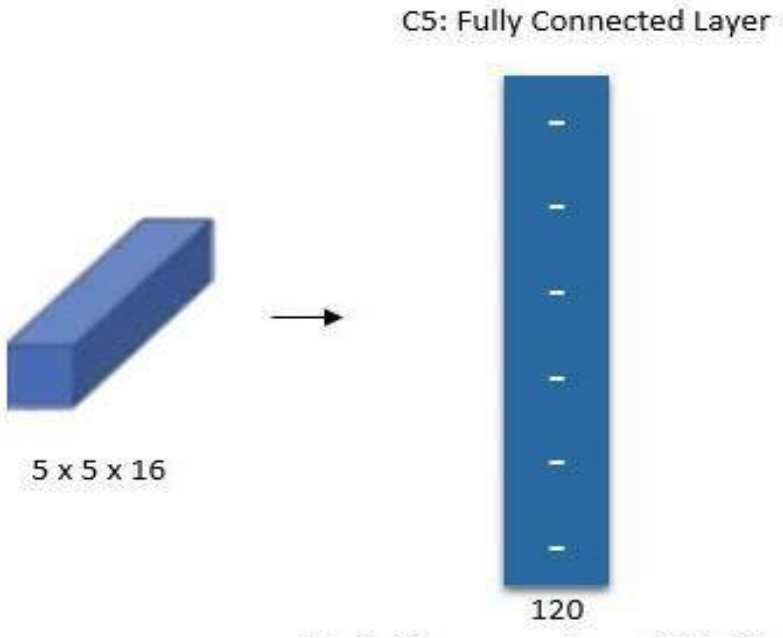
```
trainable params = (weight + bias) * feature maps
                 = (1 + 1) * 16 = 32
```

```
connections = (input + bias) * feature maps * feature map size
            = (2*2 + 1) * 16 * 5*5 = 2000
```

10.2.6. 5th Layer : Convolution layer (C5)

The fifth layer (C5) is a fully connected convolutional layer with 120 feature maps each of size 1×1.

Each of the 120 units in C5 is connected to all the 400 nodes (5x5x16) in the 4th layer S4.

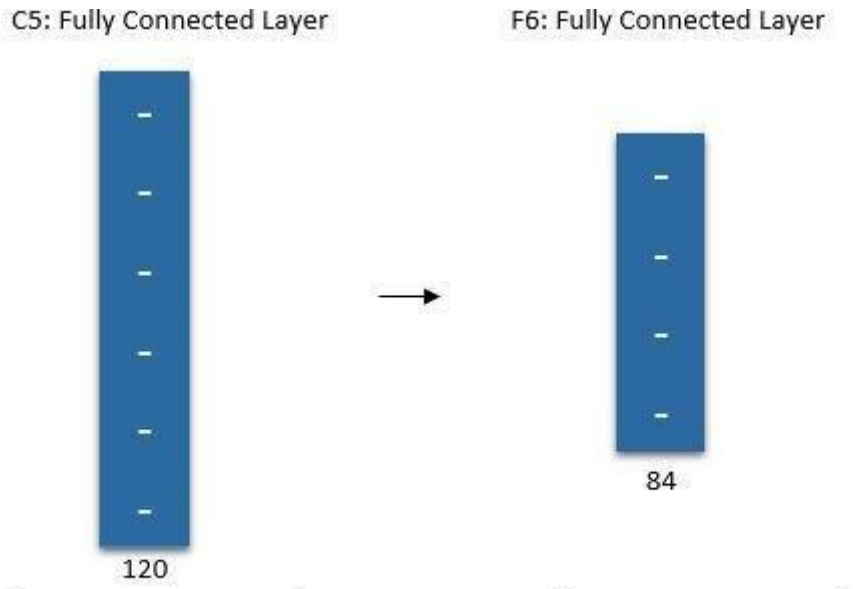



```
trainable params = (weight * input maps + bias) * feature maps
                 = (5 * 5 * 16 + 1) * 120 = 48120
```

```
connections = (input + bias) * feature maps * feature map size
            = trainable params * feature map size
            = 48120 * 1 * 1 = 48120
```

10.2.7. 6th Layer : Fully Connected layer (F6)

This layer is just a simple neural network layer with 84 output neurons.

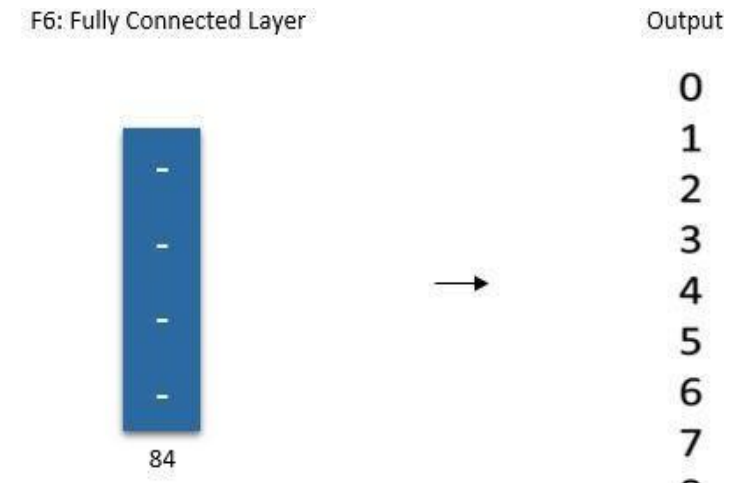



```
trainable params = connections = (input + bias) * output
                                = (120 + 1) * 84 = 10164
```

10.2.8. 7th : Output layer

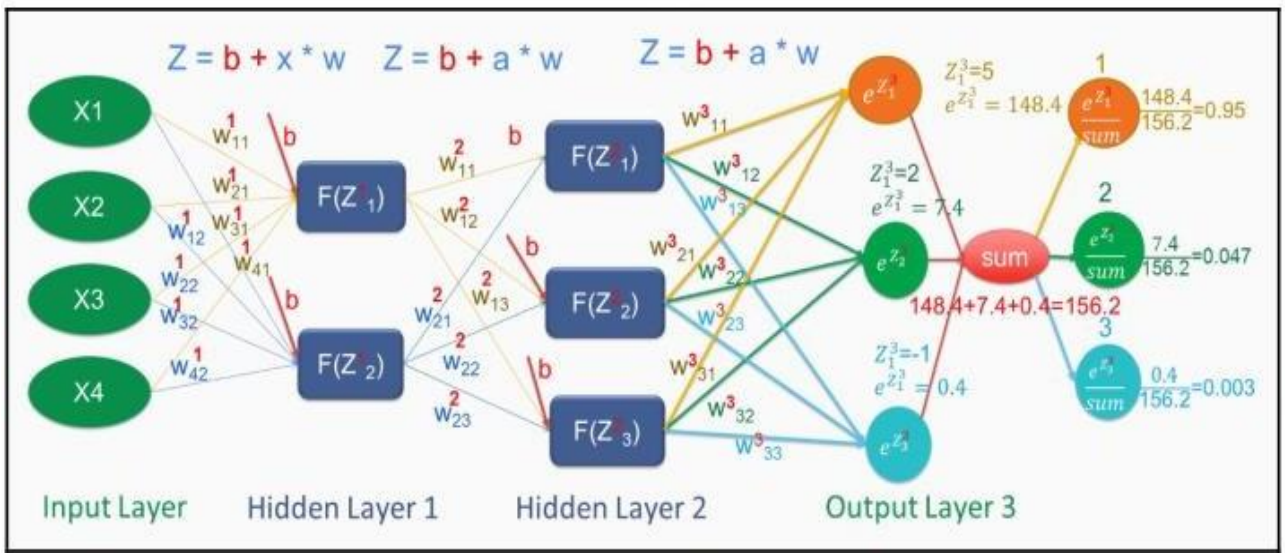
There is a fully connected softmax output layer ŷ with 10 possible values corresponding to the digits from 0 to 9. The softmax function is used in the last fully connected layer to be able to convert outputs from the previous layer into probabilities for each output class.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$



Trainable parameters = Weight + Bias
= (120 x 84) + 84 = 10164

Exemple :



$Z_1 = 5, e^{z_1^3} = 148.4.$

$Z_2 = 2, e^{z^3} = 7.4$

$Z_3 = -1, e^{z_3^3} = 0.4$

$$e^{z_1^3} + e^{z_2^3} + e^{z_3^3} = 156.2$$

Each of these values is divided by the sum to attain the final percentages. For class 1, we get 95%, class 2 gives us 4.7%, and class 3 gives us 0.3%.

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

11. Conclusion

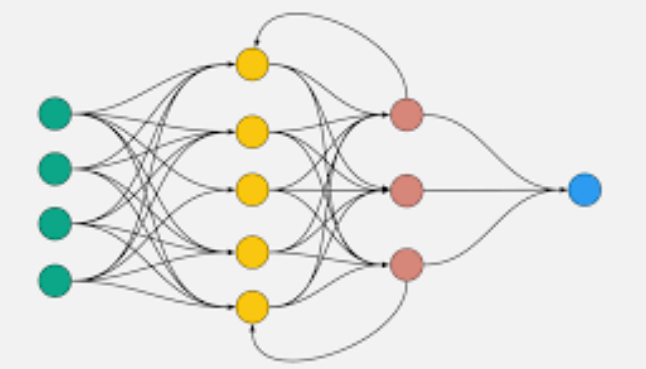
Convolutional Neural Networks represent a fundamental paradigm shift in computer vision, addressing the core challenges of visual data processing through elegant architectural innovations. By introducing spatial locality through convolution operations, computational efficiency through pooling mechanisms, and hierarchical feature learning from simple edges to complex objects, CNNs have transcended the limitations of traditional fully-connected architectures to become the cornerstone of modern computer vision.

The emergence of transfer learning has democratized access to powerful visual representations, enabling practitioners to leverage pre-trained models like LeNet-5 and VGG-16 for specialized domains with limited data. This principle has accelerated innovation across diverse fields, from medical imaging to autonomous systems, fundamentally altering the landscape of computer vision applications.

As CNNs continue to evolve and extend into new domains beyond image classification—including object detection, semantic segmentation, video analysis, and even natural language processing—the foundational principles of local connectivity, parameter sharing, and hierarchical learning remain central to ongoing developments in artificial intelligence. The convolutional approach to visual understanding will undoubtedly remain a cornerstone of AI, ensuring its continued relevance and impact for years to come.

Chapter 03

Recurrent Neural Networks



1. Introduction

Artificial neural networks have experienced remarkable growth due to their outstanding performance in numerous domains such as computer vision, speech recognition, and natural language processing. These successes are primarily attributable to architectures like multilayer perceptrons (MLP) and convolutional neural networks (CNN), which excel at processing structured data of fixed dimensions.

However, classical feedforward architectures present significant limitations when processing sequential or variable-length data. In the real world, much data exhibits temporal or sequential structure: audio signals, time series, text, videos, motion trajectories, etc. Such data require models capable of capturing temporal dependencies and memorizing contextual information.

This chapter introduces recurrent neural networks (RNNs), specifically designed to address these limitations. We will explore their architecture, learning mechanisms, advanced variants (LSTM, GRU), as well as their positioning in the current deep learning landscape, particularly in relation to Transformers.

2. Feedforward Architectures: Principles and Limitations

2.1. Definition and Operating Principle

Feedforward neural networks are models in which information flows in only one direction: from input layers to output layers, without any loops or feedback. This unidirectional propagation ensures that each neuron receives signals exclusively from neurons in the previous layer.

Typical architectures: Multilayer perceptrons (MLP) and convolutional neural networks (CNN) belong to this category. In an MLP, information propagates through fully connected layers, while in a CNN, convolutional filters extract local and hierarchical features.

2.2. Advantages of Feedforward Architectures

These architectures present several major advantages that explain their widespread adoption in numerous domains:

- **Excellent performance on structured data:** MLPs and CNNs achieve state-of-the-art results on numerous classification and regression benchmarks.
- **Speed during inference:** Once trained, these models can make predictions very quickly through optimized and parallelizable matrix computations.

- **High-dimensional data processing capability:** CNNs in particular excel at processing high-resolution images through their hierarchical architecture.
- **Automatic feature extraction:** Deep layers automatically learn abstract and discriminative representations, eliminating the need for manual feature engineering.
- **Efficient learning algorithm:** Gradient backpropagation, combined with modern optimizers (Adam, RMSprop), enables efficient and stable learning.
- **Massive parallelization:** Matrix operations can be distributed across GPUs and TPUs, considerably accelerating training.

2.3 Fundamental Limitations Related to Data Size

A fundamental drawback of feedforward networks is the requirement for fixed input and output sizes for all training examples. This architectural constraint stems directly from the use of fixed-dimension weight matrices.

Concrete example: In handwritten digit recognition using the MNIST dataset, all images must have exactly the same size (28×28 pixels). Any dimensional variation requires prior resizing, which can result in information loss or distortion. In many real-world applications, this constraint is problematic:

- **Audio signals and speech:** Utterance duration varies considerably from one sentence to another, making direct use of feedforward networks difficult.
- **Handwriting:** The length of handwritten words and sentences varies, requiring adaptive approaches.
- **Texts and documents:** Customer reviews, news articles, or legal documents have extremely variable lengths (from a few words to several thousand).
- **Machine translation:** Source text length can differ significantly from target text length, making fixed-dimension architectures impossible to use.
- **Time series:** Sensor, medical, or financial data have variable observation durations depending on context.

2.4 Absence of Contextual Memory

Beyond dimensional constraints, feedforward networks process each input independently, without memorizing previous inputs. This poses a major problem for tasks requiring contextual or temporal understanding.

Practical consequence: A feedforward network analyzing words in a sentence treats them as isolated entities, thus losing all information about order and grammatical context. For example, the sentences 'the cat eats the mouse' and 'the mouse eats the cat' would be treated identically if we only consider vocabulary, although their meanings are radically different.

3. Sequential Data and Temporal Dependencies

3.1 Nature of Sequential Data

A temporal or ordinal dimension characterizes sequential data where the order of elements carries information. Unlike independent and identically distributed data, elements of a sequence exhibit statistical dependencies.

Examples of sequential data :

- Natural language: Words in a sentence follow grammatical and semantic rules that create short- and long-distance dependencies.
- Audio signals: Successive sound samples are correlated, forming phonemes, words, and coherent sentences.
- Financial time series: Stock prices depend on their history and exhibit temporal trends, cycles, and volatilities.
- Biomedical data: ECG, EEG, or EMG signals show temporal patterns characteristic of physiological state.
- Videos: A sequence of images (frames) where each image's content depends on previous images.

3.2 The Problem of Temporal Dependencies

In signals and sequences, data are not independent: past context strongly influences interpretation of the present. This fundamental property requires models capable of capturing these dependencies.

Illustrative example: Consider the sentence 'The sky is ___'. A human can easily guess that the missing word might be 'blue', 'cloudy', or 'clear' thanks to context. This ability relies on memorization and interpretation of prior context. In contrast, a feedforward model analyzing only the previous word 'is' would have no information to make a relevant prediction.

Dependencies can be of several types:

- Short-distance dependencies: Relationships between elements close in the sequence (e.g., subject-verb agreement in a short sentence).
- Long-distance dependencies: Relationships between distant elements (e.g., anaphoric reference in a long text).
- Hierarchical dependencies: Nested syntactic structure in language or musical structure.

3.3 Sliding Window Approach: A Partial Solution

A classical solution for adapting feedforward models to sequence processing consists of using a sliding window that traverses the signal. At each temporal position, a static classifier such as an MLP or CNN analyzes the window content (a fixed number of consecutive elements).

Operating principle: Given a sequence x_1, x_2, ..., x_T, a window of size w slides along the sequence, and at each time t, the model receives as input [x_{t-w+1}, ..., x_t]. This approach allows processing of variable-length sequences by decomposing the problem into fixed-size subproblems.

Advantages:

- Compatibility with existing feedforward architectures
- Ability to process sequences of arbitrary length
- Possible parallelization of different window processing

Fundamental limitations:

- Limited memory: Context considered is limited to window size. Information beyond this window is lost.
- Local decisions: Each prediction is made independently, without considering the complete sequence history.
- Arbitrary choice: Window size must be fixed a priori, which is difficult without domain knowledge.
- Computational redundancy: Central elements of the window are reanalyzed multiple times as the window slides.
- Absence of global memory: The model cannot build a coherent representation of the entire sequence.

4. Motivation for Recurrent Neural Networks

Humans possess a remarkable ability to memorize past information to interpret the present and anticipate the future. This cognitive competence relies on short- and long-term memory, which allows maintaining and manipulating contextual information.

To artificially reproduce this ability, it is necessary to endow neural networks with a form of internal memory. This is precisely the objective of recurrent neural networks: create an architecture capable of maintaining an internal state that evolves throughout sequence processing.

Guiding principles of RNNs:

- Sequential processing: Traverse the sequence element by element, from left to right (or in natural temporal order).
- Recurrent memory: Maintain a hidden state that summarizes past information and is updated at each time step.
- Parameter sharing: Use the same weights at each time step, allowing the model to generalize independently of position in the sequence.
- Architectural flexibility: Allow variable-length inputs and outputs, as well as different operating modes (many-to-one, one-to-many, many-to-many).

5. Recurrent Neural Networks (RNN): Architecture and Operation

5.1 Definition and Basic Architecture

Recurrent neural networks (RNNs) are architectures specifically designed for processing sequences and variable-length signals. Unlike feedforward networks, RNNs possess recurrent connections that create feedback loops in the computational graph.

Fundamental principles:

- Using a sliding window to traverse the sequence step by step.
- Introduction of recurrent connections allowing reinsertion of the previous hidden state as additional input at the current time step.

- Weight sharing across time, enabling the network to learn patterns independently of their position in the sequence.

5.2 Recurrent Connections and Memory Mechanism

A recurrent connection allows reinsertion of a neuron's output (or hidden state) at time t-1 as additional input at time t. This mechanism confers a form of dynamic memory on the network that evolves throughout sequence processing.

At time t, the network takes into account:

- The current input x_t (the sequence element at time t)
- Its own hidden state at the previous time h_{t-1}, which summarizes information accumulated up to this instant

This feedback creates implicit memory allowing the network to capture temporal dependencies. The hidden state h_t acts as a compressed representation of the entire sequence past.

Characteristics of this memory :

- **Implicit:** Information is encoded in neural activations, without explicit memory structure.
- **Distributed:** Information is spread across all neurons of the hidden state.
- **Adaptive:** The network automatically learns which information to memorize and which to forget.
- **Compressed:** The hidden state has a fixed dimension, imposing compression of the complete history.

5.3 Mathematical Modeling

The dynamics of an RNN can be formalized mathematically by a system of recurrent equations describing how the hidden state and output evolve at each time step.

Fundamental RNN equations:

$$\boldsymbol{h_t = f(w_{xh} * x_t + w_{hh} * h_{\{t_1\}} + b_h)}$$

$$\boldsymbol{y_t = g(w_{hy} * h_t + b_y)}$$

Where:

- x_t: Input vector at time t (e.g., encoded word, audio sample, etc.)
- h_t: Hidden state at time t, which summarizes accumulated information
- y_t: Network output at time t (prediction, classification, etc.)
- W_{xh}: Weight matrix connecting input to hidden state
- W_{hh}: Recurrent weight matrix connecting previous hidden state to current hidden state
- W_{hy}: Weight matrix connecting hidden state to output
- b_h, b_y: Bias vectors
- f: Non-linear activation function for hidden state (typically tanh or ReLU)
- g: Output activation function (softmax for classification, linear for regression)

Interpretation: The hidden state equation shows that h_t depends on both current input x_t and previous state $h_{\{t-1\}}$. This recursive dependence allows the network to memorize information from previous time steps. Repeated composition of this transformation across time creates an information propagation path from sequence beginning to end.

5.4 Temporal Unfolding

To better understand RNN operation, it is useful to visualize the 'unfolding' of the network across time. Although the same RNN cell is reused at each time step, we can conceptually represent the computation as a chain of copies of this cell, each corresponding to an instant.

Unfolded view:

$$h_1 = f(w_{xh} * x_1 + w_{hh} * h_0 \ + b_h) \rightarrow h_2 = f(w_{xh} * x_2 + w_{hh} * h_1 \ + b_h) \rightarrow h_3 = f(w_{xh} * x_3 + w_{hh} * h_2 \ + b_h) \rightarrow \dots$$

This representation highlights that the same weights (w_{xh}, w_{hh}) are shared across all time steps, and that information from sequence beginning progressively influences all subsequent steps.

5.5 Learning and Backpropagation Through Time (BPTT)

RNN learning relies on a generalization of the gradient backpropagation algorithm called Backpropagation Through Time (BPTT). This method treats the unfolded network as a very deep feedforward network, where each time step corresponds to a virtual layer.

BPTT principle :

1. **Forward pass:** Sequentially calculate hidden states h_1, h_2, ..., h_t and outputs y_1, y_2, ..., y_T from inputs x_1, x_2, ..., x_t.
2. **Loss calculation:** Evaluate the cost function L over the entire sequence (or certain time steps, depending on task).
3. **Backward pass:** Calculate loss gradients with respect to weights by going back through time, from last time step T to first.
4. **Weight update:** Use an optimizer (SGD, Adam, etc.) to adjust weights w_{xh}, w_{hh}, w_{hy} based on calculated gradients.

Recurrent gradient calculation: The loss gradient with respect to hidden state h_t depends not only on gradient at time t, but also on gradient at time t+1, creating recursive dependence:

$$\partial L/\partial h_t = \partial L/\partial y_t * \partial y_t/\partial h_t + \partial L/\partial h_{\{t+1\}} * \partial h_{\{t+1\}}/\partial h_t$$

5.6 Vanishing Gradient Problem

During backpropagation through time, gradients are repeatedly multiplied by recurrent weights w_{hh} and activation function derivative. If these multiplicative terms have a norm less than 1, gradients decrease exponentially as they propagate toward first time steps.

Mathematical analysis: The loss gradient with respect to distant hidden state h_1 involves the product of T-1 Jacobian matrices $\partial h_{\{t+1\}}/\partial h_t$. If the spectral norm (maximum eigenvalue) of these matrices is strictly less than 1, the product tends toward zero exponentially fast.

Practical consequences :

- Inability to learn long dependencies: The network cannot capture relationships between distant events in the sequence (e.g., grammatical agreement in long sentences).

- Slow or stagnant learning: Weights are practically not updated for early sequence portions.
- Bias toward short dependencies: The model favors memorization of local patterns over global context.

5.7 Exploding Gradient Problem

Conversely, if Jacobian matrix eigenvalues are greater than 1, gradients increase exponentially, causing numerical instability.

Consequences:

- **Learning divergence:** Weights take very large values, causing numerical overflows.
- **Oscillations:** The cost function oscillates violently without converging.
- **NaN (Not a Number):** Calculations produce invalid values, stopping training.

Common solution - Gradient Clipping: A simple and effective technique consists of limiting gradient norm. If $||\nabla|| > \text{threshold}$, then $\nabla \leftarrow (\text{threshold} / ||\nabla||) * \nabla$. This prevents excessive updates while preserving gradient direction.

6. RNN Architectures Combined with Classical Layers

In practice, recurrent layers are rarely used alone. They are generally combined with other types of layers (dense, convolutional, attention) to create powerful hybrid architectures.

6.1 Typical Architecture

General scheme :

- **Input layers:** Preprocessing or feature extraction (embedding for text, convolutions for audio signals or images).
- **Recurrent layers:** One or more RNN layers to model temporal dependencies and build enriched sequential representation.
- **Output layers:** Dense (fully connected) layers for classification, regression, or sequence generation.

6.2 RNN Operating Modes

RNNs can be configured in different modes depending on task nature:

- Many-to-one: One input sequence, one unique output (sentiment classification, activity recognition).
- One-to-many: One unique input, one output sequence (image caption generation, music generation).
- Many-to-many (synchronized): Input and output sequences of same length, synchronized (sequence labeling, frame-wise speech recognition).
- Many-to-many (encoder-decoder): Variable-length input sequence, potentially different-length output sequence (machine translation, text summarization).

6.3 Stacking RNN Layers (Stacked RNN)

To increase representational capacity, it is common to stack multiple RNN layers. The first layer output becomes the second layer input, and so on. This allows the network to learn hierarchical representations at different abstraction levels.

Advantages :

- Richer and more abstract representations
- Better capacity to capture complex patterns
- Performance improvement on difficult tasks

Disadvantages :

- Significant increase in parameter count
- Longer training and inference times
- Increased overfitting risk
- Amplified gradient problems

7. RNN Applications

Recurrent neural networks are widely used in numerous domains where data exhibit sequential or temporal structure.

7.1 Speech Recognition

RNNs model temporal dependencies in audio signals to transcribe speech to text. Modern systems (such as early Google Voice Search) used bidirectional RNNs and LSTMs to capture both past and future context.

7.2 Handwriting Recognition

Online handwriting (with stylus) produces coordinate sequences (x, y) over time. RNNs can learn to recognize characters and words from these temporal trajectories.

7.3 Natural Language Processing (NLP)

RNNs revolutionized NLP before the advent of Transformers:

- **Machine translation:** Encoder-decoder architectures to translate from one language to another.
- **Automatic summarization:** Condensation of long documents into concise summaries.
- **Sentiment analysis:** Classification of opinion expressed in text.
- **Language modeling:** Prediction of next word in a sequence.
- **Named Entity Recognition (NER):** Identification of named entities in text.

7.4 Time Series Analysis

RNNs excel in time series prediction and analysis:

- **Finance:** Stock price prediction, anomaly detection in transactions.
- **Biomedical:** ECG, EEG signal analysis, epileptic seizure prediction.
- **IoT sensors:** Failure prediction, predictive maintenance.
- **Meteorology:** Weather forecasting based on measurement history.

7.5 Content Generation

- **Text generation:** Automatic creation of poems, stories, dialogues.
- **Musical composition:** Generation of melodies and harmonies.
- **Speech synthesis:** Conversion of text to natural speech.

8. In-Depth Limitations of Classical RNNs

Despite their major conceptual contribution and success in numerous applications, classical RNNs present several theoretical and practical limitations that restrict their effectiveness in complex real-world contexts.

8.1 Short-Term Memory and Information Compression

RNNs store past information in a fixed-dimension hidden state. This constraint imposes implicit compression of the entire sequence history into a limited-size vector. As the sequence lengthens, this compression becomes increasingly lossy.

Consequence: Old information is progressively overwritten by new information. In a long sequence, the network 'forgets' events from the beginning, even if they are crucial for the final decision.

8.2 Strictly Sequential Processing and Absence of Parallelism

The recurrent nature of RNNs imposes strictly sequential computation: h_t can only be calculated after $h_{\{t-1\}}$. This temporal dependence prevents any form of parallelization within a sequence.

Major limitation: On modern GPUs/TPUs designed for massive parallel computation, this sequential constraint becomes a critical bottleneck. Training on long sequences or large corpora becomes prohibitively slow.

8.3 Difficulty of Interpretation and Lack of Explainability

The implicit memory of RNNs, encoded in distributed activations of the hidden state, makes analysis of internal behavior difficult. It is complex to determine which specific information the network chose to memorize or forget.

Implications :

- Difficulty debugging model errors
- Lack of transparency in critical applications (medical, legal)
- Impossibility of formally verifying behaviour

8.4 Sensitivity to Initialization and Presentation Order

RNNs are sensitive to weight initialization and sequence presentation order during training. Small variations can lead to significantly different performance.

9. Gated Architectures: Long Short-Term Memory (LSTM)

9.1 Theoretical Motivation and History

LSTM (Long Short-Term Memory) networks, introduced by Hochreiter and Schmidhuber in 1997, were specifically designed to solve the vanishing gradient problem and enable learning of long-term dependencies. The revolutionary idea is to create a quasi-linear gradient propagation path through time.

9.2 Memory Cell Principle

An LSTM cell introduces an explicit memory state (cell state) C_t that can be maintained, modified, or erased in a controlled manner through a gating system. This mechanism allows fine control of information flow.

The three main gates:

1. **Forget Gate:** Decides which information from previous memory state C_{t-1} should be forgotten. It calculates by using the following formula:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

2. **Input Gate:** Decides which new information should be added to memory state. It consists of two parts:

   $i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$ and $\tilde{C}_t = \tanh(W_C * [h_{t-1}, x_t] + b_C)$

3. **Output Gate:** Decides which information from memory state is exposed to next layer: $o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$ and $h_t = o_t * \tanh(C_t)$

4. **Memory state update:**

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

   Where $\odot$ denotes element-wise product (Hadamard product). This equation shows that memory state is a weighted combination of old state (partially forgotten) and new information (partially integrated).

9.3 LSTM Advantages

- Efficient learning of long dependencies: Quasi-linear memory state path allows gradients to propagate efficiently over long sequences.
- Increased numerical stability: Sigmoid gates (values between 0 and 1) allow smooth information flow control, reducing gradient explosion problems.
- Selective memory: Network automatically learns which information is relevant to memorize and which can be forgotten.
- Wide industrial adoption: LSTMs became the de facto standard for many sequence processing applications before the Transformer era.
- Architectural flexibility: LSTMs can be combined with other architectures (CNN, attention) for powerful hybrid models.

9.4 LSTM Limitations

- Complex architecture: Three gates and additional memory state significantly increase parameter count (approximately 4x compared to simple RNN).
- High computational cost: Additional gate calculations slow down training and inference.
- Optimization difficulty on very long sequences: While better than classical RNNs, LSTMs remain limited on sequences of several thousand elements.
- Sequential processing: Like RNNs, LSTMs cannot parallelize temporal processing.
- Overfitting: Large parameter count requires more data and regularization.

10. Gated Recurrent Units (GRU): Simplified Alternative

10.1 Motivation and Design

GRUs (Gated Recurrent Units), proposed by Cho et al. in 2014, were designed as a simpler and more efficient alternative to LSTMs, while retaining their main advantages. The objective was to reduce architectural complexity while maintaining the capacity to model long dependencies.

10.2 Architecture and Mechanism

GRUs merge hidden state and memory state into a single vector, and use only two gates instead of three:

1. Update Gate: Controls proportion of old information to keep and new information to integrate

2. Reset Gate: Decides which part of previous state should be used to calculate new candidate state

10.3 GRU Advantages

- **Fewer parameters:** Approximately 25% fewer parameters compared to LSTMs, reducing overfitting risk.
- **Faster training:** Fewer computations per time step significantly accelerate training.
- **Good performance/complexity tradeoff:** On many tasks, GRUs achieve comparable performance to LSTMs with fewer resources.
- **Elegant architecture:** Simplification makes model easier to understand and implement.
- **Memory efficiency:** Fewer intermediate tensors to store during training.

10.4 GRU Limitations

- **Less expressive than LSTMs:** For some very complex tasks requiring sophisticated memory, LSTMs may outperform GRUs.
- **Empirical choice:** There is no clear theoretical rule for choosing between GRU and LSTM; choice is often made through experimental validation.
- **Same parallelization limitation:** Like all RNNs, GRUs suffer from mandatory sequential processing.

10.5 LSTM vs GRU

We use LSTMs when :

- Task requires very sophisticated and selective memory
- Plenty of training data available
- Maximum performance prioritized over efficiency. We use GRUs when :

- Computational resources are limited
- Limited training data (overfitting risk)
- Training time is an important constraint
- Simplicity and interpretability are valued

11. Advanced Applications and Specialized Use Cases

11.1 Neural Machine Translation

Neural translation systems use encoder-decoder architecture: an LSTM/GRU encodes source sentence into vector representation (thought vector), then a second LSTM/GRU decodes this representation to generate target sentence. Addition of attention mechanisms greatly improved performance.

11.2 End-to-End Speech Recognition

Modern speech recognition systems combine CNNs for acoustic feature extraction and bidirectional LSTMs for temporal modeling. CTC (Connectionist Temporal Classification) architectures enable automatic alignment between audio and transcription.

11.3 Multivariate Time Series Prediction

In financial, meteorological, or energy domains, LSTMs can simultaneously process multiple correlated time series to improve predictions. For example, predicting electricity consumption while accounting for temperature, time, day of week, etc.

11.4 Conditional and Controlled Generation

RNNs can be conditioned on specific attributes to generate controlled content: text generation in particular style, musical composition in given genre, voice synthesis with specific emotional characteristics.

12. RNN vs Transformers: In-Depth Comparison and Paradigm Shift

12.1 Motivation for Transition to Transformers

Despite improvements brought by LSTMs and GRUs, certain fundamental limitations of recurrent architectures persist:

- **Strict sequential dependence:** Computing h_t necessarily requires h_{t-1}, preventing complete parallelism.
- **Linear temporal cost:** Computation time grows linearly with sequence length, making very long sequence processing prohibitive.
- **Persistent difficulty with very long dependencies:** Despite gating mechanisms, capturing dependencies over several hundred or thousand elements remains difficult.
- **Context bottleneck:** Fixed-dimension hidden state limits amount of information that can be transmitted over long distances.

These limitations motivated a fundamental paradigm shift: completely abandon recurrence in favor of global attention mechanism. Transformers, introduced by Vaswani et al. in revolutionary paper 'Attention is All You Need' (2017), are based on idea that any temporal dependency can be modeled by attention, without requiring recursive memory.

12.2 Fundamental Transformer Principle

Unlike RNNs that process sequences element by element, Transformers:

- **Process entire sequence in parallel:** All elements are processed simultaneously, fully exploiting GPU/TPU capabilities.
- **Use self-attention mechanism:** Each element can directly 'look at' all other sequence elements, enabling capture of arbitrarily long dependencies in constant time.
- **Have no recurrent connections:** Architecture is purely feedforward with attention, eliminating gradient problems linked to recurrence.

Self-Attention: For each sequence element, calculate weighted sum of all elements, where attention weights are learned to reflect relative importance of each element. Mathematically: Attention(Q, K, V) = softmax(QK^T / √d_k)V

12.3 Positional Encoding

A crucial point is that Transformers, by their purely attention-based nature, have no intrinsic notion of element order or position in sequence. To address this limitation, position information is explicitly injected via positional encodings.

Common approaches :

- Sinusoidal encodings: PE(pos, 2i) = sin(pos / 10000^(2i/d)) and PE(pos, 2i+1) = cos(pos / 10000^(2i/d))
- Learned positional embeddings: Position vectors trained jointly with model.

12.4 Case Study: Natural Language Processing

RNN/LSTM (2013-2017): LSTM-based sequence-to-sequence architectures dominated NLP: machine translation (Google Neural Machine Translation), language modeling, text generation. These models enabled significant advances but remained limited by sequence length and training time.

Transformer (2017-present): Transformers dominate NLP today with models like BERT (encoder), GPT (decoder), T5 (encoder-decoder). These models far surpass RNNs in accuracy, generalization capacity, and training time thanks to massive parallelization on very large corpora.

Empirical results: On standard benchmarks (GLUE, SQuAD, WMT), Transformers achieve state-of-the-art scores with gains of 5-15% compared to best RNN/LSTM models, while training several times faster.

12.5 When to Still Use RNNs?

Despite Transformer dominance in NLP and many domains, RNNs remain relevant and sometimes preferable in certain specific contexts:

- **Short sequences or real-time applications:** For sequences of a few dozen elements, RNN/LSTM/GRU can be more efficient than Transformers. In embedded or real-time applications (smartphone voice recognition, input prediction), Transformers' quadratic complexity can be prohibitive.

- **Strong hardware constraints:** On resource-limited devices (IoT, edge computing), RNNs require less memory than Transformers (no T×T attention matrix).
- **Continuous time series:** For processing continuous streams (biomedical signals, sensor data), RNNs can process incrementally without recalculating entire sequence, unlike Transformers requiring complete window.
- **Streaming inference:** RNNs can produce predictions as sequence arrives (online prediction), while Transformers generally require complete sequence.
- **Extremely long sequences:** Paradoxically, for sequences of several tens of thousands of elements, Transformers' $O(T^2)$ complexity becomes impractical, and RNN variants may be preferred (though efficient Transformers like Linformer, Performer exist).

13. Conclusion

Recurrent neural networks constitute a major conceptual advance for sequential data processing. By integrating internal memory through recurrent connections, they enabled effective modeling of temporal dependencies and overcame structural limitations of classical feedforward architectures.

Evolution of RNNs toward gated architectures (LSTM, GRU) represented decisive progress by partially solving vanishing gradient problem and enabling learning of long-term dependencies. These architectures dominated sequence processing for nearly a decade and enabled major advances in speech recognition, machine translation, and time series analysis.

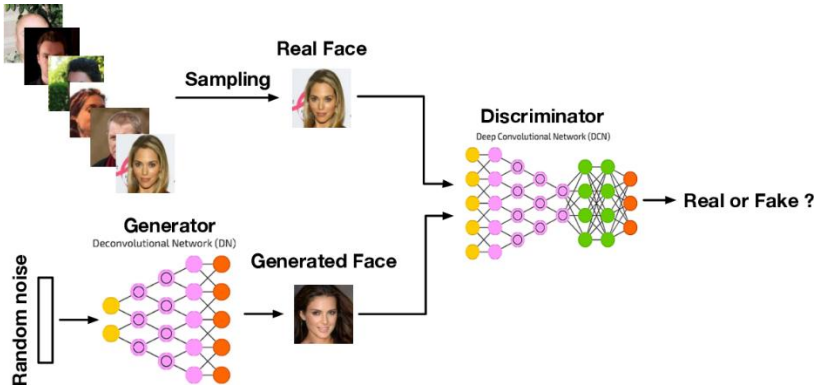
However, intrinsic RNN limitations – sequential processing, limited parallelization, difficulty with very long sequences – motivated emergence of new generation of attention-based architectures, culminating in Transformers. This paradigm shift revolutionized natural language processing and progressively extends to other domains (vision, audio, biology).

Nevertheless, RNNs retain their relevance in specific contexts: embedded applications, streaming processing, short sequences, and resource constraints. Deep understanding of RNNs also remains essential for grasping conceptual foundations of sequential deep learning and for developing future hybrid architectures.

Ultimately, RNNs form an indispensable conceptual foundation for modern artificial intelligence. They illustrate how introduction of memory and recurrence mechanisms enables capture of temporal data structure, paving way for today's and tomorrow's more sophisticated architectures

Chapter 04

Generative Adversarial Networks



1. Introduction

Generative Adversarial Networks (GANs) constitute a major advancement in the field of generative modeling in deep learning. Introduced by Goodfellow et al. in 2014, GANs are based on an adversarial learning paradigm in which two neural networks are trained simultaneously with opposing objectives. This competitive process enables learning of complex, high-dimensional data distributions without requiring explicit likelihood estimation. From a pedagogical perspective, GANs illustrate a paradigm shift in machine learning: rather than directly optimizing a single objective function, learning emerges from the interaction between two agents. This chapter aims to provide both an intuitive and formal understanding of GANs, progressing gradually from conceptual foundations to mathematical formulation and practical considerations.

The power of GANs lies in their ability to generate synthetic data of remarkable quality, thus opening numerous applications in computer vision, signal processing, and beyond. However, their training remains notoriously difficult, requiring deep understanding of their underlying mechanisms.

2. Generative Modeling: Context

2.1 Discriminative versus Generative Models

The distinction between discriminative and generative models is fundamental in machine learning:

Discriminative models: These models aim to learn conditional probabilities of the form $p(y|x)$, where y represents a label or output, and x represents input data. They focus on the decision boundary between different classes. Examples: logistic regression, SVM, neural networks for classification.

Generative models: These models seek to learn the underlying data distribution $p(x)$ or the joint distribution $p(x,y)$. They enable generation of new data instances, simulation of scenarios, and learning of rich representations. Examples: Gaussian Mixture Models (GMM), Variational Autoencoders (VAE), GANs.

Generative modeling offers several strategic advantages:

- Synthesis of realistic data for dataset augmentation
- Learning of meaningful latent representations
- Scenario simulation for sensitivity analysis
- Anomaly detection through density estimation

2.2 Motivation for GANs

Traditional generative models (e.g., Gaussian mixture models, VAEs) often rely on explicit likelihood estimation, which can be computationally expensive or restrictive in its assumptions. For example:

- VAEs require variational approximation that can lead to blurry samples
- Autoregressive models are time-consuming for sequential generation
- Flow-based models require constraining reversible architectures

GANs bypass explicit likelihood calculation by learning through adversarial training. This approach offers several advantages:

- Generation of high-quality, realistic samples
- No strong architectural constraints on the generator
- Fast parallel generation (unlike autoregressive models)
- Flexibility in choice of latent space

3. GAN Architecture

A GAN consists of two main components trained simultaneously in an adversarial framework:

3.1 The Generator (G)

The generator is a neural network that transforms a latent vector z, sampled from a simple prior distribution (typically Gaussian N(0,I) or uniform U[-1,1]), into a synthetic data sample.

Formally: G : Z → X, where Z is the latent space and X is the data space.

Generator objective: Produce samples G(z) that are indistinguishable from real data by the discriminator. The generator seeks to minimize the discriminator's ability to distinguish real data from generated data.

Typical architecture: For image generation, the generator often uses transposed convolution layers (deconvolution) to progressively increase spatial resolution, starting from a low-dimensional latent vector toward a high-dimensional image.

3.2 The Discriminator (D)

The discriminator is a binary classifier that receives either real samples from the training dataset or synthetic samples produced by the generator. It outputs a probability indicating whether the input is real.

Formally : D : X → [0,1], where D(x) represents the probability that x is a real sample.

Discriminator objective : Maximize its ability to correctly distinguish real samples (by assigning D(x) ≈ 1) from generated samples (by assigning D(G(z)) ≈ 0).

Typical architecture: For images, the discriminator generally uses standard convolutional layers to extract hierarchical features, followed by fully connected layers leading to a sigmoid output.

3.3 Training Dynamics

During training, both networks iteratively improve through their adversarial interaction:

- The discriminator is trained to better distinguish real from fake samples
- The generator learns to produce increasingly realistic samples to fool the discriminator

This continuous competition continues until equilibrium is reached, ideally when the generator produces samples indistinguishable from real data.

4. Adversarial Training Framework

4.1 Minimax Objective Function

GAN training is formulated as a minimax optimization problem between generator G and discriminator D. The value function V(D,G) is defined by:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]$$

The discriminator aims to maximize its ability to correctly classify real and generated samples, while the generator aims to minimize this objective by producing samples that the discriminator classifies as real.

4.2 Optimal Discriminator

For a fixed generator G, it is possible to analytically derive the optimal discriminator. The optimal solution is given by:

$$D(x) = p_{data}(x) / (p_{data}(x) + p_g(x))$$

where p_data is the real data distribution and p_g is the distribution induced by the generator. By substituting this optimal discriminator into the value function, it can be shown that GAN training implicitly minimizes the Jensen-Shannon divergence between the real data distribution and the generator distribution.

4.3 Game-Theoretic Interpretation

GAN training corresponds to a two-player zero-sum game. At Nash equilibrium, the following conditions are satisfied:

- The generator perfectly reproduces the data distribution: $p_g = p_{data}$
- The discriminator can no longer distinguish real from generated samples: $D(x) = 1/2$ for all x
- No player can unilaterally improve their performance

In practice, reaching this equilibrium is difficult due to the non-convex nature of loss functions and challenges in simultaneously optimizing two networks.

5. GAN Training Challenges

Although GANs are conceptually elegant, their training is notoriously difficult due to several well-documented challenges in the literature.

5.1 Mode Collapse

Mode collapse occurs when the generator produces a limited set of outputs, independent of the input latent vector. Consequently, generated samples lack diversity and do not represent the complete data distribution.

Main causes:

- The generator finds a strategy to systematically fool the discriminator with only a few examples

- Lack of explicit mechanism to encourage diversity
- Imbalance in generator and discriminator capacities

5.2 Gradient Instability

Vanishing gradients: When the discriminator becomes too accurate, it provides little informative feedback to the generator. The gradient log(1-D(G(z))) becomes very small, slowing or stopping generator learning.

Exploding gradients: Conversely, excessively large gradients can cause numerical instability and divergence during training.

5.3 Absence of Convergence Guarantees

GAN training does not guarantee convergence to a stable equilibrium. The optimization dynamics can oscillate indefinitely without reaching equilibrium, diverge completely, or converge to suboptimal equilibria.

6. Stabilization Techniques

To address GAN training instability, several stabilization strategies have been proposed:

6.1 Feature Matching

Instead of directly fooling the discriminator, the generator is trained to match the feature statistics of intermediate layers of real data.

6.2 Label Smoothing

Real labels are smoothed (e.g., 1.0 → 0.9) to prevent the discriminator from becoming overconfident.

6.3 Batch Normalization

Batch normalization reduces internal covariate shift and stabilizes gradient flow through deep layers.

6.4 Spectral Normalization

This technique constrains the Lipschitz constant of the discriminator by normalizing weights by their spectral norm.

6.5 Gradient Penalty (WGAN-GP)

Introduced with Wasserstein GANs, this technique enforces Lipschitz continuity by penalizing gradient norms.

7. Major GAN Variants

7.1 Deep Convolutional GAN (DCGAN)

DCGAN introduces specific architectural constraints to improve stability: strided convolutions, batch normalization, ReLU/LeakyReLU activations.

7.2 Conditional GAN (cGAN)

Conditional GANs extend the basic framework by conditioning both generator and discriminator on auxiliary information (labels, attributes, text).

7.3 Wasserstein GAN (WGAN)

WGAN proposes using Wasserstein-1 distance instead of Jensen-Shannon divergence, offering more informative gradients and better stability.

7.4 StyleGAN

StyleGAN introduces a style-based generator architecture, enabling high-resolution image synthesis and fine, hierarchical control of generation.

8. GAN Evaluation

8.1 Qualitative Evaluation

Visual inspection remains a primary evaluation tool. High-quality GANs produce samples that are visually realistic, diverse, without obvious artifacts, and structurally coherent.

8.2 Quantitative Metrics

8.2.1 Inception Score (IS)

The Inception Score simultaneously measures sample quality and diversity using a pre-trained Inception network.

8.2.2 Fréchet Inception Distance (FID)

FID measures the distance between real and generated feature distributions. This metric correlates well with human judgment.

8.2.3 Precision and Recall

These metrics separately evaluate fidelity (precision) and coverage (recall) of the data distribution.

8.3 Metric Limitations

No single metric fully captures generative quality. Robust evaluation requires a combination of quantitative metrics and human qualitative assessment.

9. GAN Applications

9.1 Image Synthesis and Super-Resolution

- Generation of photorealistic faces
- Image resolution enhancement
- Generation of realistic scenes and landscapes
- Creation of generative art and design

9.2 Image-to-Image Translation

- Conversion of sketches to photorealistic images
- Style transfer between domains
- Image colorization
- Day/night conversion

9.3 Medical Imaging

- Medical data synthesis
- High-quality MRI and CT image reconstruction
- Organ and tumor segmentation
- Medical data anonymization

9.4 Data Augmentation

GANs can generate synthetic samples to enrich imbalanced or rare datasets.

9.5 Anomaly Detection

By learning the distribution of normal data, GANs can identify abnormal samples:

- Financial fraud detection
- Industrial quality control
- Cybersecurity

10. Ethical and Practical Considerations

10.1 Deepfakes and Disinformation

GANs can create extremely realistic manipulated videos and images, posing risks for:

- Dissemination of false information
- Identity theft
- Manipulation of public opinion

10.2 Privacy and Consent

Training GANs on personal data raises privacy questions and requires informed consent.

10.3 Responsible Use

Responsible use requires:

- Ethical dataset selection
- Development of detection mechanisms
- Implementation of watermarking
- Public education
- Establishment of appropriate legal frameworks

11. Conclusion

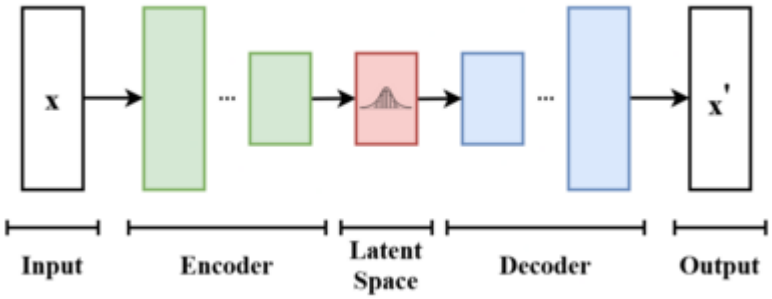
Generative Adversarial Networks represent a powerful and flexible framework for learning complex data distributions. Their adversarial learning paradigm constitutes a major conceptual innovation in machine learning.

Although their training remains a technical challenge, continuous theoretical and practical advances have considerably improved their robustness and applicability. Modern variants have demonstrated remarkable capabilities in high-resolution image generation, domain transfer, and medical data synthesis.

GANs have become an indispensable tool in modern machine learning. Nevertheless, with this power come significant ethical responsibilities. Future research directions include improving training stability, more sophisticated conditional generation, extension to new domains, and development of more robust evaluation methods.

Chapter 05

Variational Autoencoders (VAE)



1. Introduction

Variational Autoencoders (VAEs) represent a major advancement in the field of generative deep learning. Introduced by Kingma and Welling in 2013, VAEs combine the concepts of autoencoders with variational inference to create a probabilistic framework enabling both data compression and generation. Unlike classical deterministic autoencoders, VAEs learn a probabilistic latent distribution, conferring remarkable generative properties.

2. Principles of Variational Autoencoders

2.1. Fundamental Architecture

A variational autoencoder consists of two main neural networks that work in concert to learn an efficient latent representation of data:

The encoder (inference network): This network takes a data point x as input and projects it into a latent space by learning the parameters of a probability distribution. More precisely, the encoder produces two vectors: a mean vector μ and a standard deviation vector σ, which define a multivariate Gaussian distribution in the latent space. Mathematically, this is expressed as $q_\varphi(z|x)$, where φ represents the encoder network parameters.

The decoder (generative network): This network performs the inverse operation by taking a sample z from the latent space and attempting to reconstruct the original data. The decoder models the conditional distribution $p_\theta(x|z)$, where θ represents its learnable parameters. The objective is to maximize the probability of faithfully reconstructing the input data from its latent representation.

2.2. Probabilistic Theoretical Framework

The theoretical foundation of VAEs rests on variational inference and maximization of the Evidence Lower Bound (ELBO). The objective is to optimize the marginal data likelihood $p_\theta(x)$, which is generally intractable to compute directly. VAEs circumvent this difficulty by maximizing the ELBO, which constitutes a lower bound on the log-likelihood:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - KL(q_\varphi(z|x) \,||\, p(z))$$

This formulation decomposes the objective into two complementary terms. The first term, the expectation of the log reconstruction likelihood, measures the model's ability to faithfully reconstruct input data. The second term is the Kullback-Leibler (KL) divergence between the learned latent distribution $q_\varphi(z|x)$ and a prior distribution $p(z)$, typically chosen as a standard Gaussian $N(0, I)$.

2.3. The Reparameterization

A major challenge in VAE training is the stochastic sampling operation, which is inherently non-differentiable. To enable gradient backpropagation through this operation, Kingma and Welling introduced the reparameterization. Instead of directly sampling $z \sim q_\varphi(z|x)$, we sample noise $\varepsilon \sim N(0, I)$ and compute:

$$z = \mu_\varphi(x) + \sigma_\varphi(x) \odot \varepsilon$$

This reformulation separates stochasticity (ε) from learnable parameters (μ and σ), thus allowing gradients to flow through the network. The symbol $\odot$ represents the element-wise product (Hadamard product). This technique was instrumental in making VAE training feasible through stochastic gradient descent.

3. VAE for Data Generation

3.1. Generative Properties

The generative power of VAEs stems from their ability to learn a continuous and structured latent representation of data. Unlike classical autoencoders that can create 'holes' in the latent space, VAEs, through the KL regularization term, encourage a smooth and continuous latent space where interpolation between points is meaningful. This property is fundamental for generation: one can sample random points $z \sim p(z)$ from the prior distribution and decode them into new realistic data.

The structured latent space also enables meaningful vector arithmetic operations. For example, in the case of faces, one can identify directions in the latent space corresponding to semantic attributes such as age, gender, or facial expression. These properties facilitate control over the generative process and open perspectives for conditional generation and semantic editing.

3.2. Generation Process

The generation process with a trained VAE unfolds in two simple steps:

- First, we sample a latent vector z from the prior distribution p(z), typically N(0, I).
- Second, we pass this vector through the decoder to obtain the generated data $\hat{x}$ = decoder(z). The quality of generated samples crucially depends on the expressiveness of the decoder and the regularity of the learned latent space.

VAEs also offer remarkable interpolation capabilities. Taking two data points x_1 and x_2, we encode their latent representations z_1 and z_2, then linearly interpolate in the latent space:

$$z\_t = (1-t)z_1 + tz_2 \text{ for } t \in [0,1].$$

Decoding these intermediate points produces a gradual and realistic transition between the two initial data points, demonstrating the continuity of the latent space.

3.3. Limitations and Variants

Despite their advantages, standard VAEs present certain limitations. Generated samples sometimes tend to be blurry, particularly for high-resolution data such as natural images. This phenomenon is partly due to the use of L2 distance-based reconstruction terms, which penalize pixel-wise errors without consideration for human perception. Moreover, the tradeoff between reconstruction and regularization can lead to posterior collapse, where the model ignores the latent space and behaves as a simple decoder.

To address these issues, several variants have been proposed. β-VAEs introduce a coefficient β in front of the KL term to control the disentanglement-reconstruction tradeoff. Hierarchical VAEs use multiple levels of latent variables to capture structures at different scales. VQ-VAE (Vector Quantized VAE) discretize the latent space to improve reconstruction quality.

4. Applications in Data Compression and Generation

4.1. Data Compression

VAEs constitute a powerful tool for data compression through their ability to learn compact latent representations. The latent space, typically of much lower dimension than the original data, encodes the essential information necessary to reconstruct the data. This compression is particularly effective for structured data where VAEs can exploit inherent redundancies and regularities.

In the context of image compression, VAEs can achieve significant compression rates while maintaining acceptable perceptual quality. Unlike traditional compression methods like JPEG that use predefined transforms, VAEs learn directly from data an optimized representation for the task. Recent architectures like VAEs with autoregressive priors have shown competitive performance with next-generation codecs.

4.2. Image and Video Generation

Image generation constitutes one of the flagship applications of VAEs. By training on vast image corpora, VAEs learn to capture complex distributions of visual data. The models can then generate new realistic images by sampling from the latent space. Work such as NVAE (Nouveau VAE) has demonstrated that carefully designed VAE architectures can rival GANs in terms of generated image quality.

For video generation, VAEs are extended to model the temporal dimension. VideoVAEs use 3D convolutions or recurrent architectures to capture temporal dynamics. These models find applications in video synthesis, future frame prediction, and video compression. The probabilistic nature of VAEs also allows generation of multiple plausible futures for a given sequence, reflecting the inherent uncertainty in prediction.

4.3. Natural Language Processing Applications

In the field of natural language processing (NLP), VAEs have shown their utility for various tasks. For text generation, VAEs learn semantic latent representations of sentences or documents, enabling controlled text generation with desired properties. This approach facilitates tasks such as automatic paraphrasing, where the model generates reformulations by varying the latent representation.

VAEs are also used for textual style transfer, where the objective is to transform a text to adopt a certain style (formal/informal, positive/negative) while preserving semantic content. By disentangling style and content factors in the latent space, VAEs enable targeted manipulations. Other applications include dialogue completion, diverse response generation, and learning document representations for information retrieval.

4.4. Scientific and Medical Applications

VAEs find important applications in science and medicine. In genomics, they are used to learn latent representations of gene expression profiles, facilitating discovery of disease subtypes and identification of biomarkers. The VAE's ability to handle high-dimensional data with complex structures is particularly valuable in this context.

In medical imaging, VAEs serve multiple purposes: image reconstruction from undersampled acquisitions to accelerate MRI, anomaly detection by modeling the distribution of healthy images, and generation of synthetic data to address the lack of annotated data. In computational chemistry, molecular VAEs learn continuous representations of molecules, enabling optimization and discovery of new molecules with desired properties.

4.5. Anomaly Detection and Data Augmentation

Anomaly detection represents a natural application of VAEs. By training a VAE on normal data, the model learns to faithfully reconstruct such data. Anomalies, unseen during training, produce high reconstruction errors, enabling their detection. This unsupervised approach is used in cybersecurity to detect network intrusions, in predictive maintenance to identify equipment failures, and in quality control to spot defective products.

Data augmentation constitutes another important practical application. VAEs can generate realistic variations of training data, thus enriching available datasets. This technique is particularly useful when real data are scarce or expensive to obtain, such as in medical imaging or for minority classes in imbalanced problems. The controlled diversity of generated samples helps improve generalization of supervised models trained on these augmented data.

The following table summarizes the main VAE applications by domain:

Domain	Applications
Computer Vision	Image generation, compression, super-resolution, style transfer
Natural Language Processing	Text generation, paraphrasing, style transfer, dialogue systems
Medical Imaging	MRI reconstruction, anomaly detection, synthetic data generation
Genomics	Gene expression analysis, disease subtype discovery, biomarker identification
Chemistry	Molecular generation, drug discovery, property optimization
Anomaly Detection	Fraud detection, network intrusion detection, quality control

VAE vs. GAN Comparison

Criterion	**VAE (Variational Autoencoder)**	**GAN (Generative Adversarial Network)**
Theoretical Foundation	Variational inference, ELBO maximization	Game theory, minimax optimization
Sample Quality	Sometimes blurry (due to L2 reconstruction loss)	High-quality, highly realistic samples
Diversity	Good (continuous latent space)	May suffer from mode collapse
Training Stability	Stable, with a well-defined loss function	Unstable, requiring a delicate balance between generator and discriminator
Control over Generation	Excellent (latent space interpolation)	Limited (except for conditional GANs, cGANs)
Inference (Encoding)	Possible through an explicit encoder	Not possible (no encoder)
Preferred Applications	Data compression, anomaly detection, latent-space NLP	High-resolution image generation, style transfer

5. Conclusion

Variational autoencoders represent a particularly elegant class of generative models, combining theoretical rigor with practical efficiency. Their probabilistic framework offers a clear interpretation of the learned latent space and enables controlled manipulations of generated data. VAEs excel in tasks requiring both compression and generation, with a structured latent space facilitating exploration and interpolation.

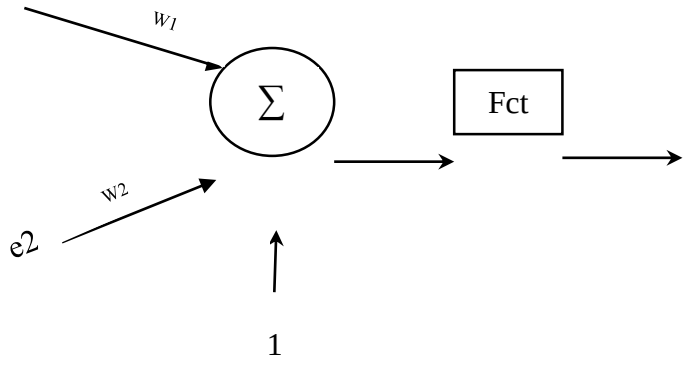
Despite certain limitations in terms of sample quality compared to adversarial approaches, VAEs continue to evolve. Recent developments such as hierarchical VAEs, VQ-VAEs, and integration with diffusion models open new perspectives. The rise of foundational generative models largely builds on principles established by VAEs, testifying to their lasting impact on the field.

The diversity of VAE applications, from image compression to drug discovery, demonstrates their versatility. As architectures become more sophisticated and computing power increases, VAEs will continue to play a central role in representation learning and generative modeling, constituting a fundamental tool for researchers and practitioners in machine learning.

Practical Exercises 01

Exercise 01:

Consider a perceptron (a single neuron) with two inputs (e1, e2) as shown schematically in the figure below:



The activation function of this perceptron is defined as follows:

$$Fct\ (x) = signe\ (x) = \begin{cases} 1 & Si\ x > 0 \\ -1 & Sinon \end{cases}$$

1- Find the values of the weights w1, w2 and the bias (or threshold) b that satisfy the following data set:

e1	e2	**d**
1	1	**1**
-1	1	**-1**
-1	-1	**-1**
1	-1	**-1**

Given that the initial values of the weights are as follows: w1 = -0.2, w2 = +0.1, b = -0.2 and the learning step η = +0.1

2- Determine the mathematical equation of the separating hyperplane that allows us to distinguish between the two classes.

3- Can the XOR function be implemented with a single perceptron? Justify your answer by explaining the limitations of a simple perceptron for this type of problem.
4- Propose a multi-perceptron model with two layers that allows for XOR, describing the role of each perceptron?
5- Show how the perceptrons are connected to each other through a diagram?

Exercise 02: Evaluating the Performance of a Neural Network

You trained a neural network to classify images into **three categories** : **Cats, Dogs, and Rabbits** . After training, you tested the model on a validation set and obtained the **following confusion matrix** :

	Predicted: Cat	**Predicted: Dog**	**Predicted: Rabbit**
Real Cat	50	5	3
Real Dog	10	60	5
Real Rabbit	2	4	40

1. Calculate the accuracy of the model.
2. Calculate the accuracy (Precision), recall (Recall) and F1 score for each class.
3. Interpret these results: is the model well balanced for all classes?

Practical Exercises 02: Generative Adversarial Networks

Exercise 1: Conceptual Questions

1. Explain the fundamental difference between generative and discriminative models. Give concrete examples of each category.
2. Describe the respective roles of the generator and discriminator in a GAN.
3. Why does GAN training not rely on explicit likelihood estimation?

Exercise 2: Mathematical Analysis

1. Formally derive the optimal discriminator D*(x) for a fixed generator G.
2. Explain the relationship between GAN training and Jensen-Shannon divergence.
3. Discuss why Wasserstein distance improves training stability.

Exercise 3: Practical Reasoning

1. Propose at least three concrete strategies to mitigate mode collapse.
2. Compare GANs and VAEs in terms of sample quality and diversity.
3. Discuss the tradeoffs between discriminator capacity and convergence.

Exercise 4: Advanced Questions

1. Explain why loss values in GANs do not necessarily correlate with perceptual quality.
2. Discuss the impact of discriminator capacity on convergence.
3. Compare adversarial learning with maximum likelihood estimation.
4. Explore the concept of 'disentanglement' in latent space.

Practical Exercises 03: Recurrent Neural Networks

Exercise 1: Fundamental Conceptual Questions

1. Explain why classical feedforward architectures (MLP, CNN) cannot effectively process variable-length sequences. What specific architectural constraints pose problems?
2. Describe the role of recurrent connections in an RNN. How do these connections enable creation of implicit memory of the past?
3. What is 'temporal unfolding' of an RNN? Why is this representation useful for understanding BPTT algorithm?
4. Compare sliding window approach with RNNs for sequence processing. What are advantages and disadvantages of each approach?

Exercise 2: Mathematical Analysis and BPTT

1. Given a simple RNN with: $h_t = \tanh(W_{xh} * x_t + W_{hh} * h_{t-1} + b_h)$. Formally derive gradient backpropagation equation $\partial L/\partial h_t$ as function of $\partial L/\partial h_{t+1}$ and $\partial L/\partial y_t$.
2. Mathematically explain why vanishing gradient occurs in RNNs. Show that if spectral norm of W_{hh} is less than 1, gradients decrease exponentially.
3. Calculate number of parameters in LSTM compared to simple RNN. If RNN has d_h hidden neurons and d_x input dimensions, how many additional parameters does LSTM require?

Exercise 3: LSTM and Gating Mechanisms

1. Explain role of each of three gates in LSTM cell: forget gate, input gate, output gate. Give concrete example where each gate plays crucial role.
2. Why can LSTMs learn longer dependencies than classical RNNs? Explain concept of 'quasi-linear gradient propagation path.'
3. Analyze memory state update equation: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$. How does this equation allow fine control of memorized information?

Exercise 4: LSTM vs GRU Comparison

1. Compare LSTM and GRU architectures. What are main differences in terms of gate number and memory mechanisms?
2. In what practical contexts would you prefer GRU to LSTM? Conversely, when would LSTM be preferable?
3. Discuss complexity/performance tradeoff between simple RNN, GRU, and LSTM. How to choose appropriate architecture for given problem?

Exercise 5: RNN vs Transformers

1. Explain why Transformers can process sequences in parallel while RNNs cannot. What is consequence of this difference on computational efficiency?
2. Compare computational complexity of RNNs ($O(T)$) and Transformers ($O(T^2)$). In what contexts is each approach more advantageous?
3. Discuss positional encoding in Transformers. Why is it necessary while RNNs don't need it?
4. Identify three applications where RNNs remain preferable to Transformers despite current dominance of latter.

Exercise 7: Advanced Questions

1. Explain gradient clipping problem. How does this technique help stabilize RNN training?
2. Discuss different RNN operating modes: many-to-one, one-to-many, many-to-many. Give application example for each mode.
3. What is bidirectional RNN? Why is this architecture useful for certain tasks and impractical for others?
4. Analyze encoder-decoder architecture for machine translation. How do attention mechanisms improve this basic architecture?
5. Discuss regularization strategies specific to RNNs: variational dropout, recurrent dropout, weight tying. How do they differ from standard dropout?

Practical Work: Image classification based on artificial neural networks

Objective of the practical work

This lab involves developing an image classification model using machine learning algorithms based on neural networks (CNNs or MLPs). You will explore the concepts of training, evaluating, optimizing, and saving models. A graphical user interface will be created to test the model, and a PowerPoint presentation detailing the steps will be submitted.

Work to be done

1. Installing and preparing the Python environment

Install the necessary libraries: cv2, keras, matplotlib, scikit-learn, tkinter, etc.

2. Downloading the dataset

3. Creation and training of a classification model:

Option 1: Implement a CNN to classify the images. Use convolutional and pooling layers.

Option 2: Implement an MLP (Multilayer Perceptron) by vectorizing the images.

In both cases, add techniques to avoid overfitting, such as:

• Dropout layers to reduce the dependency between neurons.

• Early stopping to automatically stop training when performance on the validation game stabilizes.

4. Performance Analysis

• Generate and save learning curves (loss/accuracy for training and the validation).

• Create a confusion matrix to evaluate the predictions.

• Analyze the results (frequent errors, poorly classified classes).

5. Saving and loading the model

• Implement saving the model after training in .h5 format, and loading it in order to test it on images independently.

6. Graphical User Interface (GUI)

• Develop a user interface with Tkinter or any other graphical framework.

• The user interface must allow the following:

• Load a local image.

• Display the model prediction with the associated probability.

• Display the correct class for validation (if provided).

7. Creation of a PowerPoint presentation

Create a PowerPoint presentation that includes:

• A description of the dataset and preprocessing.

• The design of CNN/MLP models.

• Learning curves.

• The confusion matrix.

• The challenges encountered and the solutions implemented.

8. Submission of work

Place all the following files in a compressed folder and send it to me before 21/03/2026 (final deadline).

• The Python source code.

• The saved template (.h5).

• Learning curves.

• The confusion matrix.

• The PowerPoint present

References

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning* (pp. 214-223).

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks,* 5(2), 157-166.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., & Bengio, S. (2016). Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning* (pp. 10-21).

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1724-1734).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems* (pp. 2672-2680).

Graves, A., Mohamed, A., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 6645-6649).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems* (pp. 5767-5777).

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., & Lerchner, A. (2017). β-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4401-4410).

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097-1105).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*.

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning* (pp. 1278-1286).

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85-117.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems* (pp. 3104-3112).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Vahdat, A., & Kautz, J. (2020). NVAE: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, 33, 19667-19679.

Van Den Oord, A., & Vinyals, O. (2017). Neural discrete representation learning. In *Advances in Neural Information Processing Systems* (pp. 6306-6315).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).