Introduction to Convolutional Neural Networks (CNNs) Classification, Object Detection, Video Recognition, and Segmentation

Ayan Chatterjee, Ph.D.

Scientist, Department of Digital Technology, NILU Kjeller
Adjunct Assoc. Prof. Department of Information Technology, Kristiania University College, Oslo

February 18, 2025

Overview

1. Artificial Neural Networks (ANNs) - 30 mins
2. Important Components - 30 mins
3. Convolution Neural Networks (CNNs) - 60 mins
4. Image Classification - 40 mins
5. Object Identification from Images and Videos - 40 mins
6. Image Segmentation - 40 mins
7. Practical Session
8. Conclusion

Neural Networks and Machine Learning

- **Machine Learning (ML)** is a subset of artificial intelligence (AI)
  - enables computers to learn patterns from data and make decisions without being explicitly programmed
  - involves training models using algorithms that improve performance based on experience
- A **Neural Network (NN)** is a specific type of machine learning model inspired by the way biological neurons in the human brain process information
  - composed of interconnected nodes, or artificial neurons, organized into layers that transform input data through weighted connections
  - used in machine learning to recognize patterns, perform classification tasks, and make predictions
  - can include different types of architectures (MLP, CNN, RNN, Autoencoder etc.)
  - can include convolutional, recurrent, or fully connected layers
  - varies based on the architecture (can be simple or deep)

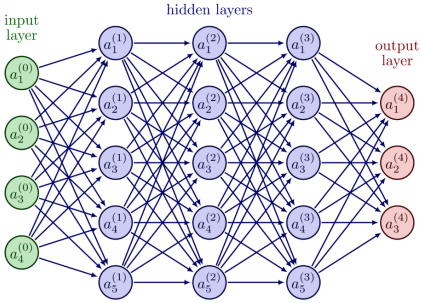
NNs vs ML

- Feature Extraction - Automatic feature engineering where ML requires manual feature engineering
- Data Requirement - Large datasets required where ML can work with small datasets
- Scalability - Handles large datasets well where ML struggles with large datasets
- Performance on Images/Speech - Excels in deep learning tasks where ML is less effective
- Training Time - Requires large computational power where ML is generally faster
- Computational Power - High where ML is moderate
- Interpretability - Low (black box models) where ML is highly interpretable

Neural Network Components

- Input Layer: Receives the raw data inputs and passes them to the next layer
- Hidden Layers: Intermediate layers that perform feature extraction and transformations using activation functions.
- Output Layer: Produces the final predictions or classifications.
- Weights & Biases: Parameters that adjust the strength of connections between neurons.
- Activation Functions: Non-linear functions (e.g., ReLU, Sigmoid, Tanh) that help the network learn complex patterns.
- Backpropagation & Optimization: Algorithms like gradient descent that adjust weights to minimize errors.

Neural Network Diagram

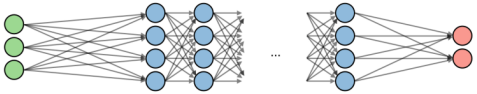


- Deep learning models are based on ANNs that contain multiple hidden layers
- Popular architectures
  - **Convolutional Neural Networks (CNNs)** – Used for image recognition and processing
  - **Recurrent Neural Networks (RNNs)** – Used for sequence data like speech and text.
  - **Transformers (e.g., BERT, GPT)** – Used for NLP tasks.
  - **Generative Adversarial Networks (GANs)** – Used for generating synthetic data.
  - **Autoencoders** – Used for unsupervised learning and anomaly detection.

Multilayer Perceptron (MLP)

- An MLP is a type of ANN composed of multiple layers of neurons
- It consists of an input layer, one or more hidden layers, and an output layer
- Each neuron in a layer is fully connected to every neuron in the next layer
- Key characteristics of MLP:
  - It uses **forward propagation** for prediction and **backpropagation** for learning
  - Activation functions to introduce non-linearity
  - It is trained using **gradient descent** and optimization techniques such as **Adam**
- Cons of MLP:
  - MLPs require large amounts of data
  - Computational inefficiency with high-dimensional images
  - Lack of spatial awareness
  - Leads to introduction of CNNs

MLP Diagram



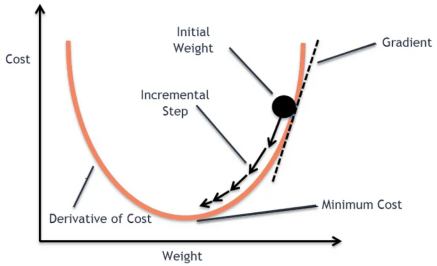
By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$\boxed{z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}}$$

where we note w, b, z the weight, bias and output respectively.

- MLP treats each pixel as an independent feature, leading to an explosion in the number of parameters
  - Example: A **28x28 grayscale image (MNIST)** has **784 input neurons**.
  - For larger images like **224x224 RGB (ImageNet)**, this means **150,528 neurons** $\rightarrow$ leading to billions of weights.
- CNNs **use shared filters (kernels)** instead of **fully connected layers**, dramatically **reducing** the number of parameters.
  - Instead of treating each pixel **individually**, CNNs **learn spatial patterns**.

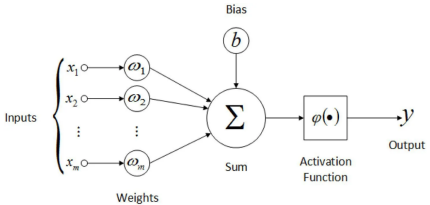
Important Component - Gradient Descent



- A partial derivative of a function, representing the rate of change
  - It tells us **how much the loss function (error) changes** with respect to each weight in the network
  - The goal of training is to **minimize the loss** by adjusting weights in the direction of the negative gradient (gradient descent)
  - It tells us the **slope** of the loss function, guiding how weights should be updated

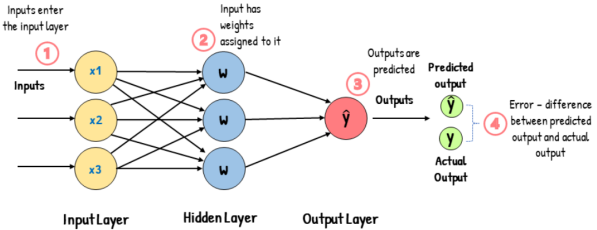
Mathematically, if L is the loss function and w represents weights, then the gradient is: $\frac{\partial L}{\partial w}$

Important Component - Forward Propagation (Prediction Process)



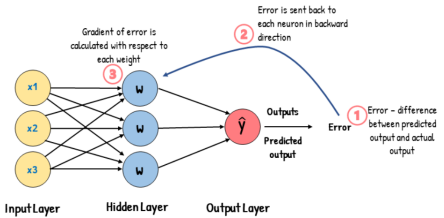
- Forward propagation is the process of moving input data through the network to make a prediction
  - Input Layer: The input features X are fed into the network
  - Hidden Layers: The input is passed through multiple layers, each having neurons with weights and activation functions.
  - Weights & Bias: Each neuron performs a weighted sum: $Z = (W * X) + B$
  - Weights and biases are the core learnable parameters that determine how a model makes predictions. Weights (W) defines the strength of the connection between neurons. Biases (B) allows the network to shift the activation threshold (up or down).
  - The network adjusts weights during training using gradient descent to minimize loss.

Important Component - Loss



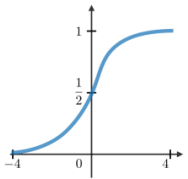
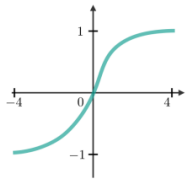
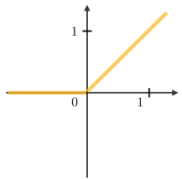
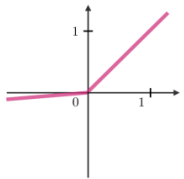
- Once the output is generated, we compare it with the actual value to compute the error (loss).
- The smaller the loss, the better the model's predictions!
  - The Mean Squared Error (MSE) is commonly used for regression tasks and is defined as: $L = \frac{1}{N}\sum(Y - \hat{Y})^2$
  - Binary Cross-Entropy (also known as Log Loss) is used for binary classification: $L = -\left[Y \log(\hat{Y}) + (1 - Y) \log(1 - \hat{Y})\right]$
  - For multi-class classification, we use Categorical Cross-Entropy: $L = -\sum Y \log(\hat{Y})$

Important Component - Back Propagation (Learning Process)



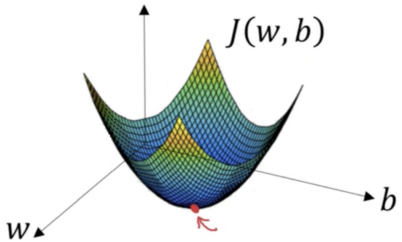
- Backpropagation is the learning process where the network adjusts its weights to reduce errors
  - Calculate Error (Loss Function)
  - Compute Gradient (Partial Derivative of Loss w.r.t. Weights): This tells us how much each weight contributed to the error. Using the chain rule, we compute: $\frac{\partial L}{\partial W}$
  - Update Weights (Gradient Descent): Adjust the weights in the opposite direction of the gradient: $W = W - \alpha \cdot \frac{\partial L}{\partial W}$ where:
    - α = learning rate (small step size to avoid overshooting)
    - $\frac{\partial L}{\partial W}$ = gradient
  - The network adjusts weights during training using gradient descent to minimize loss.
  - The process repeats for multiple epochs until the loss is minimized.

Important Component - Activation Functions

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

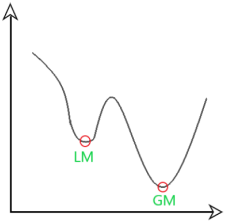
- Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model
- Without them, the network behaves like a linear model, limiting its learning power
  - ReLU prevents vanishing gradient, commonly used in hidden layers.
  - Sigmoid outputs probabilities, used in binary classification.
  - Softmax Converts logits into probability distributions (multiclass).

Important Component - Optimization (Making Learning Efficient)



- Gradient Descent is used to optimize learning
  - Batch Gradient Descent: Uses the entire dataset per update (slow but accurate).
  - Stochastic Gradient Descent (SGD): Updates after each sample (fast but noisy).
  - Mini-Batch Gradient Descent: Uses small batches (balance between speed & accuracy).

Important Component - Popular Optimizers Using Adaptive Learning Rate



- AdaGrad: Adapts learning rate per feature and decays learning rate over time.
- RMSprop: Balances AdaGrad by controlling updates and uses moving average of squared gradients.
- Adam: It combines Momentum + RMSprop and self-adjusts learning rate based on past updates.
- AdaDelta: It removes manual learning rate tuning, and uses ratio of past gradients
- Adam optimizer is commonly used for fast and stable training, and RMSProp work well for recurrent neural networks.

Important Component - Learning Rate, Dopout, and Hyperparameter Tuning

- Learning rate: It indicates at which pace the weights get updated. This can be fixed or adaptively changed.
  - If too **high** $\rightarrow$ Model oscillates and never converges
  - If too **low** $\rightarrow$ Model converges too slowly or gets stuck in a local minimum
  - Adaptive learning rate solves it where learning rate automatically adjusts during training to improve convergence and stability
- Dropout: It meant at preventing overfitting the training data by dropping out units in a neural network
  - Neurons are either dropped with probability p or kept with probability $(1 - p)$
- Hyperparameter Tuning: It is a process of selecting the best hyperparameters to optimize a deep learning model's performance.
  - Model Architecture - layers, neurons, activation functions, dropout rate
  - Training: learning rate, batch size, number of epochs, optimizer
  - Regularization: L1/L2 (weight decay with adding penalty), Dropout, Early stopping

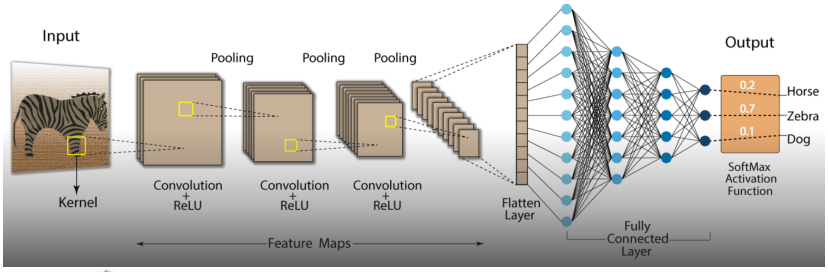
Why Convolution?

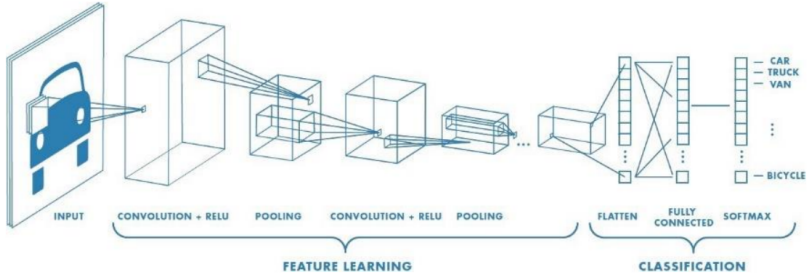
- Spatial hierarchies of features
- Better performance in image-related tasks
- Parameter sharing
- Example: Consider a color image 1000 * 1000 * 3 = 3 million number of pixels
  - Total no. of first layer weights is 3 million
  - If first hidden layer consist 1000 no. of neurons then in a FCNN the no. of input weights in 1st hidden layer will be 3 billion
  - Convolution understands the co-relation between pixels and space.
    - Pixels are nearby in space are much more correlated than those are farther space.
  - CNNs drastically reduce the number of parameters
    - Main way they do this is using layers that look like convolutions
    - **Sparse connectivity**, i.e. each neuron is connected only to an area of the input, not the whole.
    - **Weight sharing**, i.e. *similar* connections end up having the same weights. This is usually visualized as the same filter traversing the image.

CNN

- Classic architecture of a convolutional neural network
- A class of deep learning models specifically designed for processing structured grid-like data, such as images
- Inspired by the visual cortex of animals
- CNNs use a hierarchical structure that learns spatial hierarchies of features automatically from input images
- CNNs became widely known after the breakthrough of AlexNet in 2012, which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a significant margin
- CNNs have become the backbone of many computer vision tasks, such as image classification, object detection, segmentation, and even video analysis.
- CNNs are designed to handle structured grid data, but they can be applied in different dimensions depending on the data type:
  - **1D CNN (CNN1D)**: Used for sequence data like time series, audio, and text.
  - **2D CNN (CNN2D)**: Used for spatial data like images and videos.

CNN Architecture

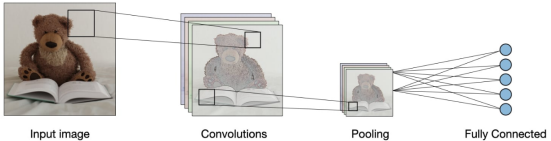




History of CNNs

- LeNet-5 (1998): Designed by Yann LeCun, LeNet-5 was one of the first CNN architectures used for digit recognition.
- AlexNet (2012): Introduced by Alex Krizhevsky, it won the ImageNet competition and popularized deep CNNs.
- VGGNet (2014): Introduced by Oxford's Visual Geometry Group, VGGNet improved upon AlexNet by using smaller 3×3 filters and deeper networks.
- GoogLeNet/Inception (2014-2015): Introduced the "Inception Module," which allowed networks to learn multi-scale features efficiently.
- ResNet (2015): Introduced residual learning, which enabled training extremely deep networks (up to 1000 layers) by mitigating vanishing gradient problems.
- DenseNet (2017): Introduced dense connections, ensuring better gradient flow and reducing parameter redundancy.
- EfficientNet (2019): Designed by Google, it introduced a compound scaling approach to optimize accuracy and computational efficiency.

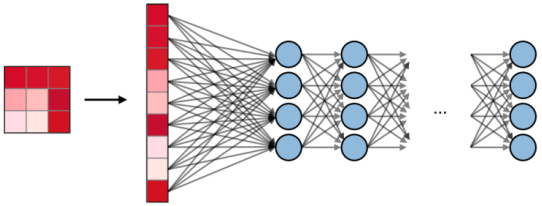
CNN Layers



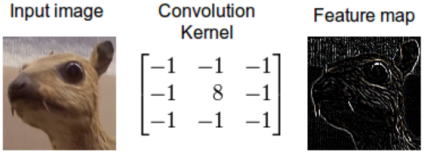
- CNNs are composed of several key layers that work together to extract features from images and classify them:
  - **Convolutional Layers**: Apply convolution operations with several different filters to detect spatial hierarchies of features, such as edges, textures, and complex shapes.
  - **Activation Functions**: Typically use ReLU (Rectified Linear Unit) to introduce non-linearity and improve training speed.
  - **Pooling Layers**: Reduce the spatial dimensions while preserving important features, commonly using max pooling.
  - **Fully Connected (FC) Layers**: Flatten the feature maps and pass them through dense layers to make final predictions.
  - **Dropout and Batch Normalization**: Techniques used to improve generalization and stabilize training.

Input Layer

- Accepts the input image in the form of (height, width, channels).
- The number of channels depends on grayscale (1 channel) or RGB images (3 channels).
- E.g., input shape = (64, 64, 3) or 64x64 RGB Image
- Defines the spatial dimensions of the image.
- Converts image data into a numerical format for processing.



Convolution - Filter, Stride, and Zero Padding



- Applies filters (kernels) to extract important features such as edges, textures, and patterns.
- Captures spatial relationships in an image.
- The kernel slides over the image, performing an element-wise multiplication and summing the results. Output $= \sum$(Input Region $\times$ Kernel) + Bias
  - **Filters**: Number of kernels used to learn different features.
  - **Kernel Size**: Size of each filter (e.g., 3x3, 5x5).
  - **Stride**: Steps by which the kernel moves over the image.
  - **Padding**: 'Same' (zero-padding to maintain dimensions) or 'Valid' (no padding).

Convolution Output Calculation

Formula for Convolution Output:

$$O_H = \frac{(H - K + 2P)}{S} + 1$$

$$O_W = \frac{(W - K + 2P)}{S} + 1$$

Example: Given

- Input Size: 6×6
- Kernel Size: 3×3
- Stride $S = 1$
- Padding $P = 1$

Calculation:

$$O_H = \frac{(6 - 3 + 2(1))}{1} + 1 = 6$$

$$O_W = \frac{(6 - 3 + 2(1))}{1} + 1 = 6$$

Final Output Size: 6×6

Importance of Padding in Convolution

Why is Padding Important?

- Maintains spatial dimensions, preventing size reduction.
- Ensures edge pixels contribute equally to feature extraction.
- Helps deep networks avoid excessive shrinking of feature maps.

How Padding Changes Dimensions:

- **No Padding (Valid Convolution)**:

$$O_H = \frac{(H-K)}{S} + 1, \quad O_W = \frac{(W-K)}{S} + 1$$

  - Output shrinks after each convolution. - Example: $6 \times 6 \rightarrow 4 \times 4$ with 3×3 kernel.

- **Padding (Same Convolution)**:

$$O_H = \frac{(H-K+2P)}{S} + 1, \quad O_W = \frac{(W-K+2P)}{S} + 1$$

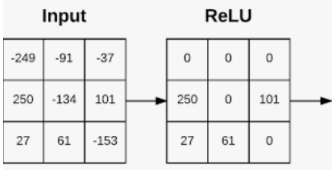
  - Maintains the original size. - Example: $6 \times 6 \rightarrow 6 \times 6$ with $P = 1$.

Padding

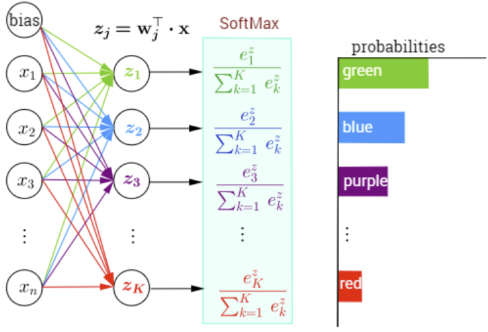
- The pixels located on the corners and the edges are used much less than those in the middle
- If we apply padding P in an input image of size W $\times$ H, the output image has dimensions (W+2P) $\times$ (H+2P)

Activation Functions

- ReLU: Removes negative values
- Sigmoid: Used for binary classification
- Softmax: Multi-class classification

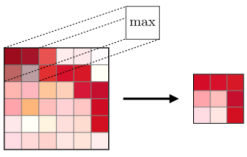
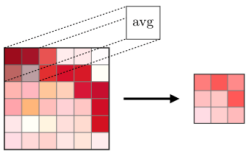


$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Pooling Layers

- Reduces spatial dimensions, a down-sampling operation, typically applied after a convolution layer
- Max pooling vs Average pooling
  - **MaxPooling** preserves edges and textures
  - **AveragePooling** is used when texture details are less important (e.g., medical images).

	Max pooling	**Average pooling**
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

Max Pooling Calculation

Formula for Max Pooling:

$$O_H = \frac{H}{S}, \quad O_W = \frac{W}{S}$$

Example: Given

- Input Size: 6×6
- Pooling Size: 2×2
- Stride $S = 2$

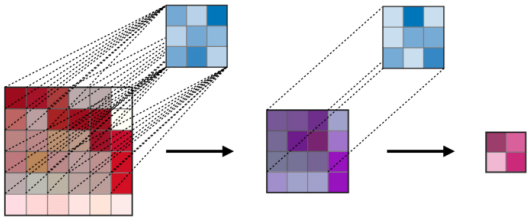
Calculation:

$$O_H = \frac{6}{2} = 3, \quad O_W = \frac{6}{2} = 3$$

Final Output Size: 3×3

Fully Connected Layers

- Converts feature maps into final predictions (Dense layer)
- High parameter count, can cause overfitting
- Dropout (Regularization) aims to help prevent overfitting by increasing testing accuracy
  - **Randomly drops** neurons during training to prevent overfitting
  - Dropout rate 0.2 - 0.3 if model is small and 0.4 - 0.5 for large models with more parameters
- Batch Normalization normalizes activations **within each batch** to improve training speed and stability.



Transfer Learning in CNN

What is Transfer Learning?

- A technique where a pre-trained model is used as a starting point for a new task.
- Helps leverage previously learned features from large datasets (e.g., ImageNet).
- Reduces training time and improves model accuracy with limited data.

Why Use Transfer Learning?

- Works well when labeled data is limited.
- Saves computational resources.
- Pre-trained models (e.g., VGG16, ResNet, DenseNet, EfficientNet) have already learned useful features.

How It Works:

1. Load a pre-trained model.
2. Freeze the earlier layers (retain general features).
3. Fine-tune the last layers on new data.

Freezing Layers in CNN2D Transfer Learning

What is Freezing Layers in CNN2D?

- Freezing layers in a pretrained CNN model means that the weights of these layers are not updated during training.
- This is useful when leveraging transfer learning, where we use a model trained on a large dataset (like ImageNet) and adapt it to a new, smaller dataset.

Why Freeze Layers?

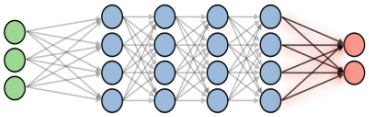
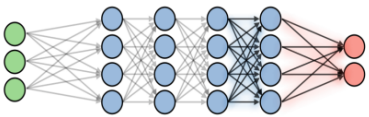
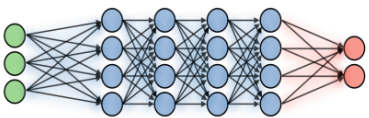
- Preserve Pretrained Features: The initial layers of CNNs detect edges, textures, and general patterns. These are universal features for any image dataset.
- Reduce Training Time: Since fewer parameters are updated, training is faster.
- Avoid Overfitting: If the new dataset is small, freezing prevents overfitting by keeping learned features intact.
- Fine-tune Higher Layers: Instead of retraining the entire model, we only train the last few layers to adapt to the new task.

Freezing Strategies

What are different strategies?

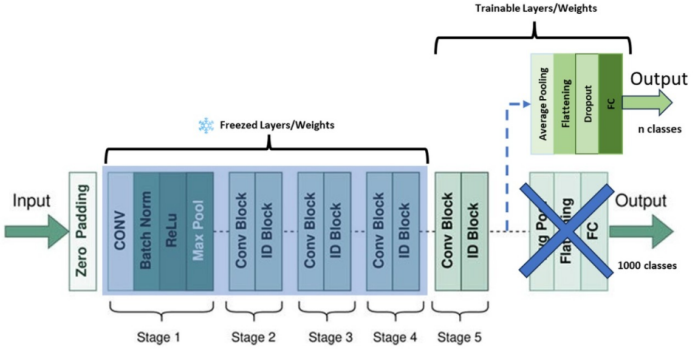
- Freeze All Layers: No layers are updated. Used for simple transfer learning.
  - Decent, but might not generalize
  - Fast
  - Small datasets, similar tasks
- Freeze Some Layers: The lower layers (feature extractors) are frozen, and only higher layers are trained.
  - Good adaptation
  - Moderate
  - Medium-sized datasets, similar but slightly different tasks
- Fine-Tuning (Unfreeze All): Initially, freeze layers, then gradually unfreeze layers to adapt to new data.
  - Best performance
  - Slow
  - Large datasets, different tasks

Transfer Learning Approaches - Illustration

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

Transfer Learning for Image Classification

- Using pretrained models (e.g., VGG16, ResNet, DenseNet, EfficientNet)
- Fine-tuning vs Feature Extraction
- Training on new datasets - binary vs multi-class
- Models - https://keras.io/api/applications/
- Follow - https://keras.io/guides/transfer_learning/



Resnet50 Model For Transfer Learning

Data Augmentation in Image Classification

Why Data Augmentation?

- Increases dataset size to prevent overfitting.
- Improves generalization for unseen images.
- Enhances model robustness to variations.

Common Data Augmentation Techniques:

- **Geometric Transformations:** Rotation, Flipping, Scaling, Cropping.
- **Color Space Changes:** Brightness, Contrast, Hue Adjustments.
- **Noise Injection:** Gaussian Noise, Blurring.
- **Random Erasing:** Hides parts of the image.

Data Augmentation

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

Object Detection in Images

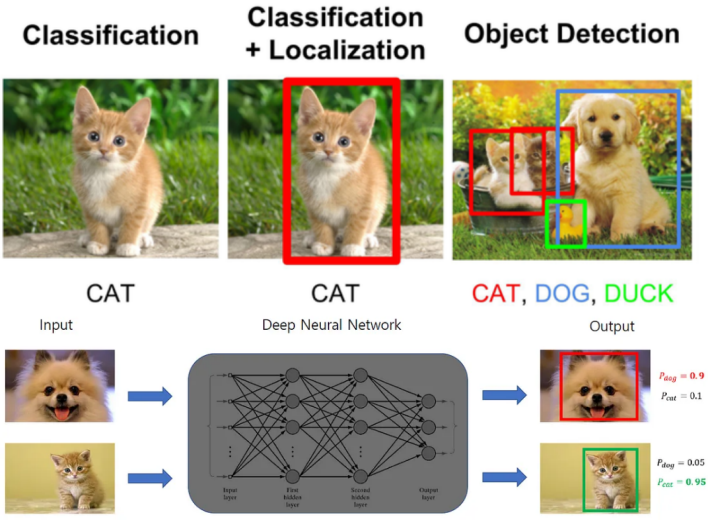
What is Object Detection?

- Object Detection algorithms act as a combination of image classification and object localization
- In Image classification, it takes an image as an input and outputs the classification label of that image with some metric (probability, loss, accuracy, etc).
- Object localization algorithm locates the presence of an object in the image and represents it with a bounding box. It takes an image as input and outputs the location of the bounding box in the form of (position, height, and width).
- Identifies and classifies objects within an image.
- Outputs bounding boxes and class labels.

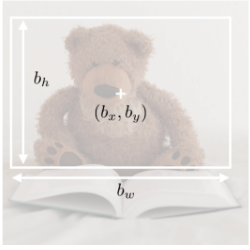
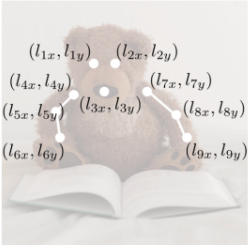
Key Applications:

- Face recognition (e.g., Face ID, security cameras)
- Autonomous vehicles (pedestrian & traffic detection)
- Medical imaging (tumor detection)

Object Detection in Images



Object Detection Methods

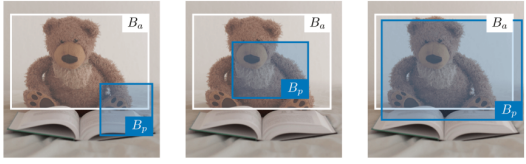
Bounding box detection	**Landmark detection**
Detects the part of the image where the object is located	- Detects a shape or characteristics of an object (e.g. eyes) - More granular
	
Box of center (b_x,b_y), height b_h and width b_w	Reference points (l_{1x},l_{1y}), ...,(l_{nx},l_{ny})

Object Detection Metrics

Intersection over Union (IoU)

- It is a function that quantifies how correctly positioned a predicted bounding box B_p is over the actual bounding box B_a.
- This metric varies from 0–1 (0–100%) with 0 implying no overlap (garbage) and 1 signifying perfectly overlapping segmentation (fat dub)
- Mean IoU : Binary (two classes) or multi-class segmentation, the mean IoU of the image is calculated by taking the IoU of each class and averaging them

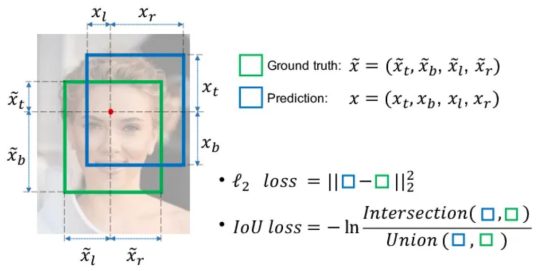
$$\text{IoU}(B_p, B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$

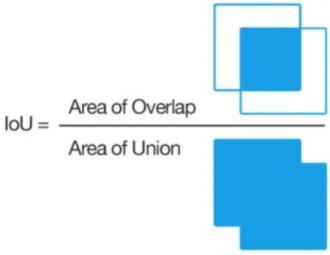


$\text{IoU}(B_p, B_a) = 0.1$ $\text{IoU}(B_p, B_a) = 0.5$ $\text{IoU}(B_p, B_a) = 0.9$

Remark: we always have $IoU \in [0,1]$. By convention, a predicted bounding box B_p is considered as being reasonably good if $IoU(B_p, B_a) \geqslant 0.5$.

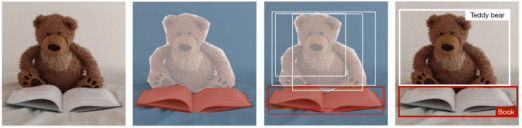
Object Detection Metrics





- Anchor boxes: They are predefined bounding boxes of different sizes and aspect ratios that help object detection models predict overlapping objects in an image
  - They are used in two-stage (Faster R-CNN) and one-stage (YOLO, SSD) object detection models.
  - Used when detecting multiple objects of different sizes
    - **Objects may overlap** (e.g., multiple cars in traffic).
    - **Objects may have different aspect ratios** (e.g., a tall person vs. a wide bus).
    - A single bounding box per grid cell is insufficient. Anchor boxes help solve this problem by assigning multiple bounding boxes per grid cell

Region with Convolutional Neural Networks (R-CNN)



Remark: although the original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.

- A two-stage object detection model that balances speed and accuracy
- It first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes
  - Backbone: A pretrained CNN (e.g., ResNet, VGG16) extracts important features from the image. (Input Image $\rightarrow$ Backbone (CNN) $\rightarrow$ Feature Maps)
  - RPN: Feature maps are passed to the Region Proposal Network (RPN) to filters out irrelevant regions by scoring their objectness (Feature Maps $\rightarrow$ RPN $\rightarrow$ Region Proposals)
  - ROI Pooling (Region of Interest): Converts region proposals into fixed-size feature maps. (Region Proposals $\rightarrow$ ROI Pooling $\rightarrow$ Fully Connected Network)
  - Fully Connected Layers (Bounding Box & Classifier): Predicts the final bounding boxes and class labels. (Final Output $\rightarrow$ Bounding Boxes + Class Labels)

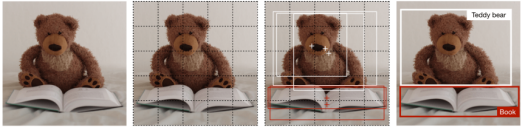
YOLO (You Only Look Once)

- YOLO is a real-time object detection algorithm that predicts bounding boxes and class probabilities in a single pass through the network.
  - Step 1: Divide the input image into a $G \times G$ grid.
  - Step 2: For each grid cell, run a CNN that predicts y of the following form:

$$y = \Big[\underbrace{p_c, b_x, b_y, b_h, b_w, c_1, c_2, ..., c_p}_{\text{repeated } k \text{ times}}, ...\Big]^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

    where p_c is the probability of detecting an object, b_x, b_y, b_h, b_w are the properties of the detected bouding box, $c_1, ..., c_p$ is a one-hot representation of which of the p classes were detected, and k is the number of anchor boxes.

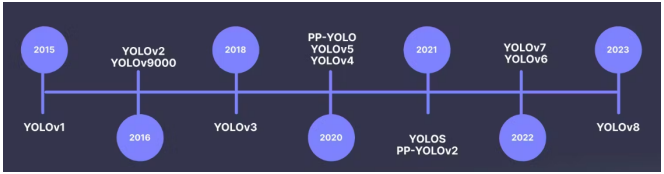
  - Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



Remark: when $p_c = 0$, then the network does not detect any object. In that case, the corresponding predictions $b_x, ..., c_p$ have to be ignored.

YOLO Architecture

- Backbone (Feature Extractor) → Extracts important features from the input image.
- Neck (Path Aggregation Network, PAN) → Helps refine feature maps.
- Head (Detection Layer) → Predicts bounding boxes & class labels.
- YOLO is the best real-time object detection model.
- YOLOv8 is the latest and most optimized version.
- Works for both images and videos.
- Can be used for real-time detection and custom training.



Key Differences Between YOLO and Faster R-CNN

Feature	Faster R-CNN	YOLO
Detection Type	Two-Stage	One-Stage
Speed (FPS)	Slower (5-10 FPS)	Faster (30-100 FPS)
Accuracy	Higher	Slightly Lower
Processing	Proposes regions before classification	Directly classifies in one pass
Best For	High-accuracy tasks	Real-time applications
Computational Cost	Higher	Lower
Real-time Suitability	No	Yes
Backbone Networks	ResNet, VGG	Darknet, CSPNet
Use Cases	Medical imaging, Autonomous driving	Surveillance, Drones, Robotics

Image Segmentation: Background

What is Image Segmentation?

- Divides an image into multiple regions (segments).
- Helps in object recognition, scene understanding, and medical imaging.
- Can be classified into:
  - **Semantic Segmentation** - Groups similar objects (e.g., all cars).
  - **Instance Segmentation** - Differentiates between objects (e.g., car 1, car 2).

Applications:

- Medical Imaging (MRI, CT Scan analysis).
- Self-driving Cars (Lane, Pedestrian Detection).
- Satellite Image Analysis (Land Cover Classification).

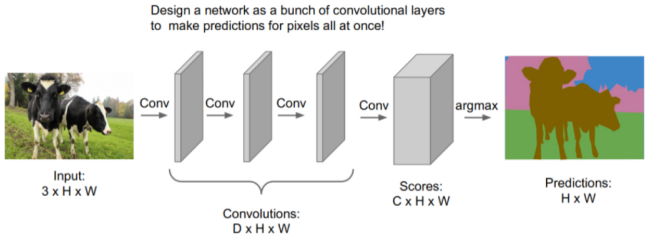
Semantic Segmentation

Definition:

- Assigns a label to each pixel in the image.
- Groups objects of the same class together.
- No distinction between individual instances.

Applications:

- Autonomous Driving - Lane and road detection.
- Medical Imaging - Tumor segmentation in MRI scans.
- Satellite Image Analysis - Land use classification.



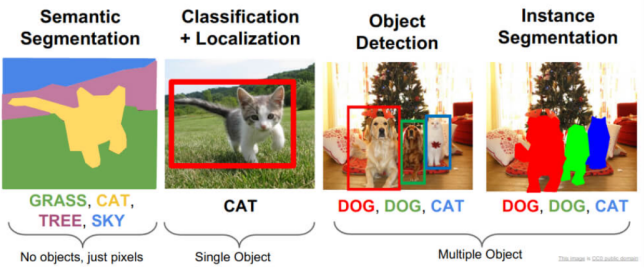
Instance Segmentation

Definition:

- Detects and segments individual objects in an image.
- Distinguishes between multiple objects of the same class.
- Combines object detection and segmentation.

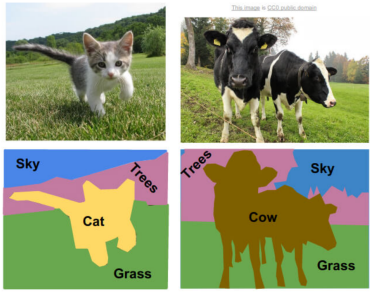
Applications:

- Autonomous Vehicles - Pedestrian and vehicle detection.
- Retail Analytics - Counting individual products in shelves.
- Augmented Reality - Object tracking for AR applications.



Differences

Feature	Semantic Segmentation	Instance Segmentation
Purpose	Assigns class label to pixels	Detects and segments each object
Differentiates Objects?	No	Yes
Example	All cars have same label	Each car has a unique label
Computational Cost	Lower	Higher



Latest Image Segmentation Methods

Popular Deep Learning Methods for Segmentation:

- **U-Net** - Encoder-Decoder structure with skip connections.
- **DeepLabV3+** - Uses atrous convolution for capturing multi-scale context.
- **Mask R-CNN** - Extends Faster R-CNN for instance segmentation.
- **PSPNet** (Pyramid Scene Parsing) - Captures global and local context.
- **SegFormer** - Transformer-based segmentation model.

Comparison of Methods:

Model	Type	Use Case
U-Net	CNN-based	Medical imaging
DeepLabV3+	CNN-based	Multi-scale segmentation
Mask R-CNN	Region-based	Object instance segmentation
PSPNet	CNN-based	Scene parsing
SegFormer	Transformer-based	High-performance segmentation

U-Net: Image Segmentation

U-Net Architecture:

- Consists of an **Encoder-Decoder** structure to predict pixel-wise classifications.
- Skip connections help retain spatial information.
- Designed for biomedical image and satellite image segmentation.

Steps in U-Net Processing:

- Downsampling (Contracting Path) - Feature extraction using CNN.
- Bottleneck - Connects encoder and decoder.
- Upsampling (Expanding Path) - Reconstructs segmented output.

U-Net Architecture and Workflow

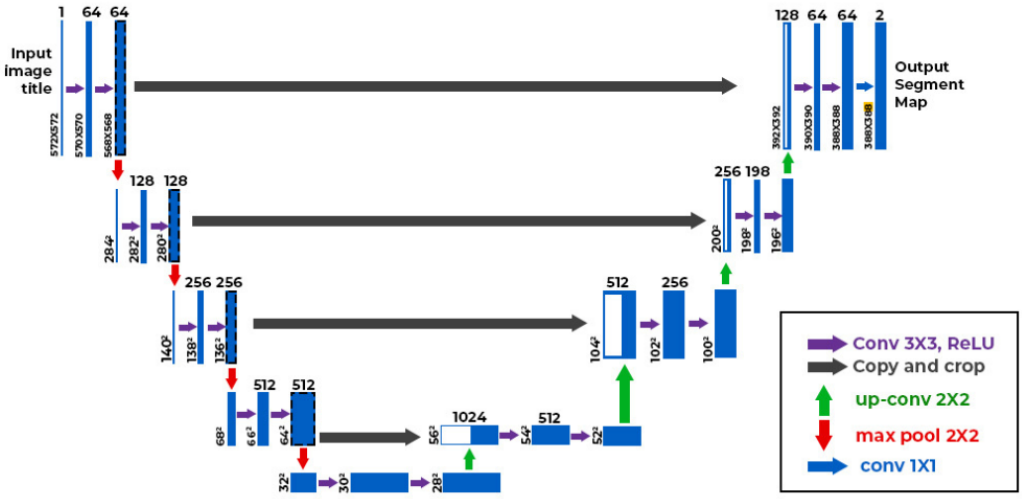
U-Net Architecture consists of:

- Encoder (Contracting Path): Extracts features using convolutional layers & max pooling.
- Bottleneck: Connects encoder and decoder, capturing high-level features.
- Decoder (Expanding Path): Upsamples features and applies skip connections to refine segmentation.

U-Net Workflow consists of:

- Input Image $\rightarrow$ Encoder (Extract Features)
- Bottleneck (Bridge between encoder & decoder)
- Decoder $\rightarrow$ Reconstructs the image with segmentation masks
- Output $\rightarrow$ Pixel-wise segmentation map

U-Net Architecture and Workflow - Diagram



Hands-on Task - Image Classification with Keras

Step 1: Install Anaconda

- Download from **https://www.anaconda.com/**
- Install and open the Anaconda Navigator

Step 2: Create a Virtual Environment for TensorFlow/Keras

- Open Anaconda Prompt and run:
- `conda create --name deep-learning python=3.8`
- `conda activate deep-learning`

Step 3: Install TensorFlow and Keras

- `pip install tensorflow keras`
- Verify installation

Step 4: Install Jupyter Notebook and Run

- `pip install jupyter`
- Start Jupyter Notebook: `jupyter notebook`

Hands-on Task - Image Classification with Keras

- Train a CNN on grayscale and RGB images
- Use a small dataset to classify objects
- Compare performance with and without transfer learning

Hands-on Task - YOLO for Object Detection with Keras

- You Only Look Once (YOLO)
- Real-time object detection
- Implementation and hands-on example

Hands-on Task - Object Detection in Videos with Keras

- Implement YOLO on a new video dataset
- Compare performance on different resolutions

Hands-on Task - Segmentation with U-Net with Keras

- Train a U-Net model for binary segmentation
- Experiment with different architectures and datasets

Summary

- CNNs are powerful for vision tasks
- Transfer learning accelerates training
- Object detection and segmentation have real-world applications

References

- Goodfellow et al., "Deep Learning"
- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition.
- Stanford Univ.
  https://github.com/afshinea/stanford-cs-230-deep-learning
- Papers with Code: https://paperswithcode.com/
- Keras.
  https://keras.io/examples/vision/image_classification_from_scratch/
- https://www.cs.ubc.ca/~schmidtm/Courses/340-F22/L33.pdf
- https://keras.io/examples/vision/yolov8/
- https://keras.io/examples/vision/oxford_pets_image_segmentation/
- https://www.geeksforgeeks.org/u-net-architecture-explained/