What is Computer Vision?

Giving computers the ability to see and understand images.

Example: Humans instantly recognize a giraffe in an image , computers need algorithms to do this.

Why Computer Vision is Important

Technology is adopted when it improves how things are done**,** such as making them:

- Faster
- Cheaper
- Automated instead of manual
- Easy instead of difficult
- Convenient instead of inconvenient
- Scalable

Applications Across Industries

Computer vision is used in almost every industry**:**

- Automotive
- Manufacturing
- Human Resources (HR)
- Insurance
- Healthcare
- Oil & Gas
- Many more

Real-World Case Studies

ADNOC (Oil & Gas)

- **Problem:** Classifying rock samples was labor-intensive.
- **Solution:** Used IBM Watson to analyze rock images.
- **Impact:** Classifies up to 25,000 images/day**,** saves geologists' time.

Knockri (HR)

- **Problem:** Shortlisting candidates was time-consuming.
- **Solution**: AI video soft skill assessment using computer vision.
- **Impact:** Quantifies soft skills, helps companies hire faster.

Applications of Computer Vision

Video & Media

- **Problem:** Finding a particular scene in a video is time-consuming — requires fast-forwarding and skipping.
- **Solution:** IBM example — tag videos with keywords based on objects **i**n each scene → makes videos searchable**.**
- **Example:** Security companies can search footage for a suspect in a blue van without manually watching hours of video.

Infrastructure & Maintenance

- **Example**: Electric towers need maintenance checks for rust and structural defects.
- **Challenges:**
    - Manually climbing towers is time-consuming & risky.
    - Flying drones near wires can be unsafe.
- **Solution with Computer Vision:**
    - Take high-res images from ground level at different angles.
    - Cut images into smaller grids.
    - Develop custom classifiers : Metal structure detector and Rust level detector
    - Classify rust in grades (e.g., Grade 1 = minimal rust → Grade 6 = severe rust).

Recent Research Highlights in Computer Vision

Better Object Detection: Facebook is improving how computers detect objects — crucial for things like self-driving cars to avoid obstacles.

Image Translation: UC Berkeley is turning images into different styles — like changing a horse to a zebra or summer to winter.

Motion Transfer: *Everybody Dance Now* lets you copy dance moves from one person and apply them to another in a video.

Brainstorming Your Own Computer Vision Application

1. **Start with the problem, not the solution.** Example: Think about what real need you want to solve.
2. **Focus on real-life tasks in different industries.** Example: medicine (cancer detection), driving (driver alertness), security (surveillance), manufacturing (quality checks), insurance (damage checks).

3. **Look for everyday small problems.** Example: finding receipts, identifying plants, tracking pests — anything that can make life or work easier.

What is a Digital Image?

A digital image is a rectangular array of numbers. Each number represents a pixel's intensity. For gray-scale images, intensity ranges from 0 (black) to 255 (white). More levels mean better image quality. Fewer levels mean lower quality, cartoon effect.

Pixel

The smallest block in an image. Pixels are arranged in rows and columns. Example: a 500×500 image has 500 rows and 500 columns. Each pixel is located by its (row, column) index.

Color Images (RGB)

Color images have three channels: Red, Green, Blue. Each channel is like its own gray-scale image. Combining channels creates the full-color image.

Note:

- Image Masks used to identify objects in an image.
- Mask pixels are usually 0 (black) or 1 (white).
- A video is a sequence of images called frames.

Image Classification:

Image Classification is the process of automatically assigning a **label (class)** to an image. For example, classifying an image as a *cat*, *dog*, *car*, *digit*, etc.

Applications of Image Classification:

- Smartphones: Organize photos by people, events, or trips.
- Medical Imaging: Helps doctors detect anomalies (unusual things)in X-rays, MRIs, or CT scans.
- Self-driving cars: Recognize traffic signs, pedestrians, and other vehicles.
- Education: Automated grading of handwritten homework (digits or letters).
- Satellite Imaging: Classify land use types like forest, water, urban areas.
- Security: Face recognition for unlocking devices or verifying identity.

Challenges:

- Changes in viewpoint (angle)
- Changes in illumination (lighting conditions)
- Deformation (shape or pose variations)
- Occlusion (object partly hidden)
- Background clutter (complex or messy backgrounds)

Image Classification with K-Nearest Neighbors (KNN)

What is KNN:
K-Nearest Neighbors (KNN) is one of the simplest supervised classification algorithms. It predicts the class of an unknown image by finding the *K* closest labeled samples in the training data and assigning the most frequent class among them.

How KNN Works:

- Images are converted into vectors (by flattening the pixel values).
- The distance (usually Euclidean distance) between the unknown sample and all training samples is calculated.
- The K closest samples are selected.
- The predicted label ($\hat{y}$) is the majority class among these neighbors.

Important Notes:

- **Data Representation:** Each image must be the same size (rows and columns). For RGB images, channels are combined into a single vector.
- **Dataset:** The data must be split into a training set (for storing labeled samples) and a testing set (for evaluating performance). A common split is 70% training, 30% testing.
- **Accuracy Calculation:** Accuracy is the ratio of correct predictions to total predictions. Often shown in a simple table comparing actual vs. predicted labels and marking correct/incorrect.
- **Choosing K (Hyperparameter):** K is not learned, it must be chosen by trying different values and checking which gives the highest accuracy on a validation set. Using only K=1 can be risky due to noise, higher K (e.g., 3 or 5) makes results more stable.
- **Adding New Classes:** It is easy to expand KNN, you only need to add new labeled samples for new categories.
- **Practical Example:** An app for cat vs. dog classification can use KNN to find the closest samples and output a class label as a result.

Limitations of KNN:

- It is slow for large datasets because it must calculate the distance to every training sample for each prediction.
- It cannot handle real-world challenges well: viewpoint changes, lighting changes, shape deformations, partial occlusion, and background clutter.
- KNN is mainly a teaching or simple demo tool; other advanced classifiers are used in practice for large-scale image recognition tasks.

Linear Classifier

What it is & Purpose:

A linear classifier is a basic model that splits data into two groups (like cat vs dog) by drawing a straight line (in 2D) or a flat plane (in higher dimensions). The goal is to check which side of

that line or plane an image falls on to decide its class.It's simple, fast, and forms the base for bigger, smarter models.

How it works

- First, an image is turned into a vector x (a long list of its pixel values).
- The model applies weights (w) to these values and adds a bias (b) to adjust the position of the line or plane.
- The main equation is:

  $$z = w \cdot x + b$$

  (Here, $w \cdot x$ means taking the dot product of the weights and the image vector).
- The value z is the result that tells you which side of the decision line/plane the image lands on:
  - If $z > 0$, the image is classified as dog (class 1).
  - If $z < 0$, it's classified as cat (class 0).
- The special line or plane where $z = 0$ is called the decision boundary, it's the border that splits the classes.

Other important points

- The raw value z can be any real number (positive or negative). To convert this into a probability, we use the sigmoid function, which looks like:

  $$\sigma(z) = 1 / (1 + e^{(-z)})$$
- The sigmoid always turns z into a number between 0 and 1:
  - If $\sigma(z) > 0.5$, we predict dog.
  - If $\sigma(z) < 0.5$, we predict cat.
- This makes the output easier to interpret, for example, $\sigma(z) = 0.9$ means there's a 90% chance the image is a dog.
- Limitation: A straight line or plane can't split classes if they overlap in weird shapes so linear classifiers don't work well on complex data.

Training & Why We Use It

Training finds the best values for the weights (**w**) and bias (**b**) so that the model can classify new images correctly. Without training, the model cannot learn to separate different classes like cats and dogs.We use training so the model can learn patterns in the data and generalize to unseen samples.

Loss & Cost Function

- A loss function measures how good or bad a single prediction is.
- If the prediction is correct, the loss is zero; if it is wrong, the loss is one.
- The cost function is the sum of all losses over the dataset**:** $\text{Cost} = \Sigma \text{Loss}(y, \hat{y})$**.**
- The cost shows how well the decision boundary works overall.

Relation Between Decision Boundary & Cost

The cost function directly depends on how well the decision boundary separates the classes. If the decision boundary cuts through clusters incorrectly, many points will be on the wrong side, increasing the cost. As you adjust **w** and **b**, the boundary shifts, changing the number of misclassified points. The ideal decision boundary perfectly separates the classes, which means the cost will reach zero.

Cross-Entropy Loss

- Cross-entropy is used instead of simple classification loss because it is smoother and easier to optimize.
- It works with the output of the logistic function and measures how well predicted probabilities match true labels.
- The cross-entropy loss is given by**:** $L(y, \hat{y}) = -[\, y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \,]$**.**
- A wrong, confident prediction produces a large loss; a confident, correct prediction produces a small loss.

Gradient Descent

Gradient descent is an algorithm used to find the values of **w** and **b** that minimize the cost function. It does this by calculating the slope (gradient) of the cost function and updating the parameters step by step in the direction that reduces the cost. The slope shows whether we should increase (if slope is +ve) or decrease (if slope is -ve) the parameter values to reach a lower cost. The update rule for the parameter is:

$$b\hat{}i=(b\hat{}i-1)-\eta\times slope(b\hat{}i-1)$$

where: b^1= new updated parameter value, 1b^i-1 = previous value ,H = learning rate, controls how big the step is and slope = the gradient of the cost at that parameter This process repeats until the cost is minimized, when the slope (gradient) becomes zero, we have reached the minimum and the parameter stops updating.

Learning Rate

The learning rate **η** controls how big each step is when updating the parameters. If the learning rate is too small, training will be very slow because the steps are tiny. If it is too large, the updates can overshoot the minimum and bounce around without settling down. That is why the learning rate is a crucial hyperparameter chosen carefully, often by testing different values and picking the one that gives the best accuracy on validation data.

How Gradient Descent Minimizes Cost

- Gradient descent starts with an initial guess for **w** and **b** and adjusts them to reduce the cost.
- With each update, the decision boundary improves and misclassifications decrease.
- This continues until the cost reaches its lowest point.

Learning Curve

A learning curve is a plot that shows the cost (or error) as a function of the number of training iterations. A good learning curve shows the cost decreasing steadily as the model learns. If the curve flattens out at a low cost, it means the model has trained well. If the curve stays high or bounces up and down, it means there might be problems with the learning rate or the training data.

What is Mini-Batch Gradient Descent

Mini-Batch Gradient Descent is an extension of standard gradient descent that lets us train models on large datasets more efficiently. Instead of using all training samples at once (like in Batch Gradient Descent), Mini-Batch Gradient Descent uses small groups (batches**)** of samples to update the parameters in each iteration.

How It Works: Each small group of samples (a mini-batch**)** calculates its own mini cost function or total loss, which is just an estimate of the true cost. The model parameters are updated after every mini-batch, not after the whole dataset, so training is faster and uses less memory.

Epochs and Iterations

When we use all samples in the dataset once, we call that one epoch. In Batch Gradient Descent, one iteration = one epoch because all samples are used at once. In Mini-Batch Gradient Descent, an epoch consists of multiple iterations, because each iteration only uses a batch of samples. Example: If we have 6 samples and use a batch size of 2:

- Iteration 1: first 2 samples
- Iteration 2: next 2 samples
- Iteration 3: last 2 samples → this completes 1 epoch
- So, for batch size = 2, 1 epoch = 3 iterations.

To find the number of iterations per epoch:

$$\text{Iterations per epoch} = \text{Total samples} / \text{Batch Size}$$

Cost and Accuracy

For each mini-batch, we calculate a noisy version of the cost function, this is less stable than using the whole dataset, but faster. At the end of each epoch, we check the accuracy on the validation data to see how well the model generalizes. If you train for too many epochs, accuracy on validation data may decrease even if training accuracy keeps increasing, this is called overfitting. Mini-Batch Gradient Descent helps reduce training time and balances efficiency with good generalization.

Argmax Function: The argmax function simply picks out the index of the largest value in a list of numbers. For example, if a list Z has [5, 10, 2], then argmax(Z) is 1, pointing to the second item. This is how the model picks the class with the strongest score.

SoftMax:

SoftMax handles multi-class problems by using one decision plane per class, where each plane calculates a score (Z value) for its class. These scores, which come from simple linear equations, are passed through the SoftMax function to turn them into smooth probabilities that always sum to 1. The argmax function then picks the class with the highest probability as the final prediction. It's called *Soft*Max because, unlike a hard yes/no split, it softens the decision by providing probabilities, showing how likely an input belongs to each class instead of just giving a single rigid label.

How Training Works: Training a SoftMax classifier is very similar to logistic regression. The model tweaks the weights so each plane correctly splits its class from the rest. The cross-entropy loss measures how well the probabilities match the true labels, lower is better.

One-vs-Rest: Build one plane for each class versus all other classes.
One-vs-One: Make planes for every possible pair of classes.
These setups are common for Support Vector Machines (SVMs) or when SoftMax doesn't fit the task well.

SVM:

SVM Purpose: Used for classification, works best when you want a clear margin between classes.
Non-Linearly Separable Data: Many datasets cannot be separated with a straight line, that's why we transform data into higher dimensions.
Kernel Trick: Kernels (like Linear, Polynomial, RBF) transform data implicitly, saves time and makes SVM powerful for non-linear data.
RBF(Radial Basis Function) Kernel is most common.
Gamma: Controls how tight or loose the decision boundary is.

- Low gamma = smooth, may underfit.
- High gamma = tight, may overfit.
- Best gamma is chosen using validation data.

Soft Margin: If perfect separation is not possible, SVM allows some misclassifications, controlled by regularization parameter C.
Validation: Use a train-validation split to pick the best Gamma and C to balance fitting and generalization.
Overfitting Risk: High gamma or low C can cause SVM to fit noise, always check on validation data.

Image Features:

Image features are meaningful measurements like color, texture, or edges that help classify images better than raw pixel values. Simple features like color histograms count pixel intensities but often ignore spatial arrangement. To get robust results, we extract features that capture relationships between pixels, making classification less sensitive to shifts or noise.

HOG (Histogram of Oriented Gradients)

HOG is a popular feature extraction method that focuses on the shape and structure of objects in an image. It works by converting the image to grayscale, then computing gradient magnitude and direction at each pixel. The image is divided into small cells, and each cell gets a histogram of gradient directions weighted by magnitude. Neighboring cells are grouped into blocks and normalized to handle lighting changes. The final HOG feature vector, combining all these histograms, is used with classifiers like SVMs for reliable object detection.

Neural Networks:

A neural network is a system that learns patterns in data by using many small units called neurons. Each neuron does a simple calculation: it takes input, applies weights, adds a bias, and passes it through an activation function like a sigmoid.
These neurons are arranged in layers**:**

- The input layer receives the data.
- The hidden layers process the data and find patterns.
- The output layer makes the final prediction.

In the cat vs dog example, a simple line cannot separate the data. So, a neural network uses multiple linear functions and activation functions to build complex decision boundaries that separate the data correctly.
A network learns by changing its weights and biases using gradient descent, so it can improve with experience. Such networks are called Feedforward Neural Networks or fully connected networks, and they can handle big, complex tasks that simple models cannot solve.

Fully Connected Neural Network Architecture

A fully connected neural network means every neuron in one layer connects to all neurons in the next. It is the basic structure for building powerful neural models for tasks like classification.

- For multiclass tasks, the output layer has one neuron per class and uses softmax to pick the class with the highest score.
- Adding more hidden layers makes the network deep, which helps learn complex patterns but increases the risk of overfitting and training difficulty.
- The ReLU activation function is preferred in hidden layers to avoid the vanishing gradient problem.
- We pick the best architecture (number of layers and neurons) by testing on validation data.
- Dropout, batch normalization, and skip connections are tricks to train deep networks better and reduce overfitting.
- Hidden layers act like kernels in SVMs, they learn features directly from raw data.
- Training is more complex than logistic regression because the loss surface is harder to optimize, so better gradient descent methods are needed.

How CNNs Build Features

CNNs build features from an image using learnable filters called kernels. These filters slide over the image to detect patterns like edges, corners, or textures. Each kernel creates an activation map that highlights where that pattern appears.

- Similar to Sobel filters in HOG features, kernels can detect edges in different directions.
- Instead of giving just one value, each kernel produces a whole feature map.
- Using multiple kernels produces multiple feature maps, which means more details are learned.
- ReLU is applied to each feature map to keep only positive values, helping the network learn non-linear patterns.

Adding Layers

By adding more convolutional layers, CNNs build deeper and more complex features. Each layer takes the output of the previous one and extracts new patterns.

- Early layers detect simple features like edges.
- Middle layers detect shapes and parts.
- Deeper layers detect objects or faces.
- Each output feature map acts like input for the next layer's kernels.
- Stacking layers makes CNNs powerful for recognizing detailed patterns.

Receptive Field

The receptive field is the area of the input image that influences one pixel in the feature map. A larger receptive field means the network sees more of the image context.

- A single convolution covers only a small patch.
- Adding more layers increases the receptive field without needing big kernels.
- Larger receptive fields help the model understand bigger parts of the image, which is important for detecting whole objects.

Pooling

Pooling layers reduce the size of feature maps while keeping important information. This helps make the network faster and less sensitive to small shifts.

- Max pooling is the most common method: it picks the largest value in a small region (like 2x2).
- Pooling reduces the number of parameters.
- It increases the receptive field because fewer pixels represent a bigger area of the original image.
- It makes the model more stable if the image shifts slightly.

Flattening and the Fully Connected Layers

After feature learning, the feature maps are flattened into a single long vector, which becomes the input for fully connected layers.

- Flattening reshapes 2D feature maps into 1D vectors.
- For example, a 7x7 output becomes a vector with 49 elements.
- If there are multiple channels, they all get flattened and stacked.
- The fully connected layers use this vector to make the final prediction.

Popular CNN Architectures and Transfer Learning

Over the years, several CNN designs have shaped how image classification is done today. Each new architecture solved specific problems and pushed accuracy higher. We also use these powerful models again through transfer learning instead of training from scratch.

LeNet-5

Yann LeCun's LeNet-5 (1989) was an early CNN mainly for reading handwritten digits (MNIST).

- Uses 5×5 filters, pooling layers, and simple fully connected layers.
- Ends with neurons (120, 84) and sigmoid activation to output predictions.

AlexNet

AlexNet brought CNNs back into the spotlight in 2012 by setting a new record on ImageNet.

- Jumped accuracy from 51% (older methods) to 63.3%.
- Uses large kernels like 11×11, many channels, so lots of parameters and high data needs.

VGGNet

VGGNet showed that stacking small 3×3 filters works better than big kernels.

- Fewer parameters but same receptive field.
- Deeper structure, popular versions are VGG-16 and VGG-19.

ResNet

When networks got too deep, gradients often vanished during training. ResNet solved this with skip connections.

- Skip paths help gradients flow easily, so very deep networks (32 layers or more) can train well.
- Residual learning is the main idea here.

Using Transfer Learning

Instead of building huge CNNs from zero, we reuse these tested models.

- Swap the last SoftMax with a new one matching your own number of classes.

- The rest of the CNN acts like a feature extractor.
- For smaller datasets, people may add an SVM instead of SoftMax.

What Is Object Detection?

Object detection goes beyond simple image classification. In classification, you identify *what* object is present in an image, such as a dog or a cat. In localization, you find *where* the object is by drawing a bounding box around it. Object detection combines both tasks but works with multiple objects at once. It detects what each object is and exactly where it is located in the image.

How Sliding Window Works

One basic method for object detection is the sliding window. In this approach, you take a small, fixed-size window and move it systematically across the image from left to right. At each position, you crop out that part of the image and run a classifier to check if it contains the object you are looking for or if it is just background. After scanning one row, the window moves down a bit and repeats the process until the whole image is covered. This helps detect objects wherever they appear in the image.

Bounding Boxes

When an object is found, its location is shown using a bounding box, which is simply a rectangle drawn around the object. This box is described by its top-left corner (x-min, y-min) and its bottom-right corner (x-max, y-max) or by using the top-left corner along with the width and height. The model's job is to predict these coordinates as accurately as possible for each detected object.

The Detection Pipeline

The complete object detection pipeline works like classification but includes bounding boxes too. A model is trained with images that have both class labels, such as "dog" or "cat", and the correct bounding box coordinates. The model learns to predict both the class and the location. When you give it a new image, it outputs what objects are present and draws a box around each one with the predicted coordinates.

Confidence Scores

An object detector also provides a score for each prediction to show how confident the model is about its result. This score ranges from 0 to 1. For example, if the model is unsure that an image shows a dog, the score might be 0.5. If it is very confident, the score could be 0.99. To avoid wrong predictions, you usually set a score threshold, like 0.9, and ignore any detections below that number. This helps reduce false positives and keeps the results more accurate.

Overview of Haar Cascade Classifiers

Haar feature-based cascade classifiers are a popular method for detecting objects like cars, traffic lights, pedestrians, and stop signs. This approach was introduced by Paul Viola and Michael Jones in 2001. It works by training a detection system with two types of images:

- **Positive images** — contain the target object
- **Negative images** — contain only background

The system learns to distinguish between the object and the background using a large number of training samples.

Haar Features and Wavelets

The method is based on Haar wavelets, which act like small filters (convolution kernels). These filters extract key visual features such as:

- Edges
- Lines
- Diagonal edges

Overlaying these filters on parts of the image helps the system spot patterns that match the shape or outline of the object you want to find.

How the Integral Image Works

To make detection fast, the algorithm uses an **integral image**. This is a simple but powerful concept:

- Each pixel in the integral image stores the sum of all pixels above and to the left of it.
- This lets the system quickly calculate the sum of pixel values for any rectangular region, which is needed to apply Haar features efficiently.

For example:

- The first pixel adds only its own value.
- The next one adds itself plus the value to its left.
- For a bigger region, you add up sums from multiple parts to get the total instantly.

With a base window size of 24×24 pixels, the integral image allows the detection system to generate over 180,000 features, too many to use directly.

Choosing the Best Features with AdaBoost

Not every feature is useful. The system needs to keep only the ones that help distinguish the object from the background. This is done using **AdaBoost**, which builds a strong classifier from many simple, weak classifiers.

How AdaBoost works:

- It gives weight to examples and weak classifiers.
- Misclassified examples get more weight, so the next classifier focuses on the hard cases.
- This repeats for many rounds until the combined classifier is strong and accurate.

In the Viola–Jones method, AdaBoost cuts down the huge number of features (about 180,000) to only a few thousand most effective ones — for example, around 6,000.

Using a Cascade for Fast Detection

The final step is arranging the detection in a **cascade of classifiers**. This speeds up the process by quickly rejecting regions that clearly don't match the target.

How the cascade works:

- The input image is divided into many small sub-windows.
- Each stage of the cascade checks if a sub-window could contain the object.
- If a region fails at any stage, it's immediately discarded.
- If it passes, it goes to the next stage for deeper checks.
- Only regions that pass all stages are accepted as containing the object.

For example, when detecting a car, the system checks parts of the image step by step, removing areas that fail early so only possible matches move forward.