# Welcome to PracticumAI’s *Computer Vision Course*. In these modules we’ll build on what we’ve learned about deep neural networks to demystify the technology that allows machines to interpret and make decisions based on visual data, much like how we think the human brain might. Each module in this course will provide hands-on experiences to create your own working models. In Module 1 we will take a journey through the significant milestones that have marked the progress in AI-driven computer vision, examine how AI has supercharged advancements in the field, and explore the nuances between different computer vision subdisciplines such as image recognition and object detection. You will also learn to distinguish computer vision from its more industry-specific counterpart, machine vision, and by the end, you'll be ready to instantiate and employ a computer vision model to make predictions.

# Module 1: Computer Vision Concepts

**Module 1 Course Objectives:**

By the end of this module, you will be able to:

1. Recall key milestones in AI-driven computer vision history.
2. Explain the impact of AI on computer vision advancements.
3. Differentiate between computer vision subdisciplines like image recognition and object detection.
4. Identify how computer vision differs from machine vision.
5. Instantiate a computer vision model from a framework and use that model to make predictions.

What is Computer Vision?

Computer vision primarily refers to the field within computer science that focuses on developing techniques enabling computers to understand and interpret visual information from the world. It involves algorithms and systems for acquiring, processing, analyzing, and understanding images and, in some cases, high-dimensional data from the real world (such as multi-modal models). The goal is to try and emulate human visual perception, though the applications can be broader. The advancements in deep learning, particularly convolutional neural networks (CNNs) and more recently Visual Transformers (ViTs), have significantly propelled the capabilities in this domain.

How is Computer Vision different from Machine Vision?

Machine vision is more application-focused and is often considered a subset of industrial automation. It pertains to the use of computer vision in industrial environments, primarily for inspection and process control. Machine vision systems are designed to perform specific, repetitive tasks such as quality assurance in manufacturing processes, where they might inspect products for defects, guide assembly robots, or track items through production lines. These systems typically involve a specific combination of hardware and software, including cameras, lighting, data acquisition, and processing units. The emphasis here is less on mimicking human vision and more on achieving reliable, accurate measurement and decision-making in a controlled environment. Machine vision is characterized by its high speed, reliability, and precision in constrained scenarios versus computer vision’s emphasis on flexibility and understanding.

## A Very Brief History of Computer Vision

Let’s quickly explore how we’ve reached today’s incredible level of speed and accuracy with computer vision tasks.

## 1950s-1970s - The Dawn of Computer Vision: Hubel and Wiesel's Pioneering Work

## Computer vision had its conceptual roots laid by David H. Hubel and Torsten Wiesel in the late 1950s and 1960s. Their groundbreaking research in neurophysiology, exploring the visual cortex of the brain, became the cornerstone for understanding how vision works. Their discovery of feature detectors in the cat's visual cortex, cells that responded specifically to edges, lines, and movement, gave us a basis for the fundamental mechanisms of visual processing. This work earned them a Nobel Prize in 1981 and laid the conceptual groundwork for later developments in computer vision.

Also, in the ‘60s Larry Roberts published the paper *“Machine Perception Of Three-Dimensional Solids”*. While Roberts is usually credited as being one of the founders of the Internet (he was the team lead on ARPANET, the technological precursor of the modern Internet), his paper on “machine perception” kickstarted research into using computers to analyze objects in images!

1970s-1990s - The Rise of Digital Image Processing

In the 1970s and 1980s, the field of computer vision began to take shape more formally, with digital image processing emerging as a key area. This era saw the development of basic techniques for image enhancement, restoration, and transformation. The advent of digital cameras and computers provided the necessary tools for researchers to experiment and develop algorithms that could interpret visual data. The focus during this period was primarily on understanding and processing static images, laying the groundwork for more complex interpretation of visual data.

Building off Hubel and Wiesel’s work, in the late ‘70s David Marr proposed a computational framework for modeling the neurological processes of sight.

1990s-2010s - The Era of Feature Detection and Machine Learning

The 1990s and early 2000s marked a significant shift towards using machine learning techniques in computer vision. Researchers began to focus on feature detection and extraction, where algorithms were developed to identify and track specific features within images. This era witnessed the development of various algorithms for facial recognition, object detection, and optical character recognition (OCR), which became crucial in various applications from security systems to data entry automation.

2010-2020 - Deep Learning Revolutionizes Computer Vision

The 2010s brought a revolutionary change with the advent of deep learning. The use of deep neural networks, especially CNNs, transformed computer vision. These networks, inspired by the neural structure of the human brain, could learn hierarchical representations of visual data, leading to breakthroughs in accuracy and performance in tasks like image classification, object detection, and segmentation. As we’ve covered in previous courses, this is the era that the image classification model AlexNet beat the ImageNet Challenge and cemented CNNs as the primary architecture for image tasks. This era saw the practical application of computer vision in autonomous vehicles, augmented reality, medical image analysis, and numerous other fields.

2020-Today - Present and Future: Expanding Horizons

## Today, computer vision continues to grow, integrating with artificial intelligence and other technological domains to create increasingly sophisticated applications. Advancements in real-time processing, 3D image reconstruction, edge computing and multi-modal models are opening new frontiers. The integration of computer vision with IoT and robotics is shaping the future in areas like smart cities, advanced manufacturing, and environmental monitoring. More recently, ViTs have been making strides towards even more accurate and efficient image computing tasks. The journey from understanding the brain's visual processing to enabling machines to 'see' and interpret the world autonomously marks a remarkable chapter in the history of technology.

## Computer Vision Neural Networks

As we mentioned above, CNNs have radically lowered the computational resources to perform tasks such as analyzing and classifying images. ViTs are based on the same architecture that enables Large Language Models and are slowly overtaking CNNs as the predominant networks for computer vision. Here is a brief overview of how these technologies work.

## CNNs

Key Components of CNNs:

1. **Convolutional Layers:** The core building blocks of CNNs. They perform a mathematical operation called convolution. This operation involves sliding a **filter** (also referred to as a **kernel**) over the input image and computing the dot product between the filter and local regions of the image.

Here's a basic illustration of a convolution operation using TensorFlow:

import tensorflow as tf

# Define a simple convolutional layer with one filter

conv\_layer = tf.keras.layers.Conv2D(filters=1, kernel\_size=(3, 3), strides=(1, 1), padding='valid')

1. Kernel Size: The area in pixels that the filter looks at. A 3 by 3 pixel filter is the standard kernel size. Larger kernels reduce computational load but potentially miss feature nuance that smaller filters may find.
2. Strides: The step size, in pixels, that the filter moves over the image. Larger strides reduce computational load but potentially miss feature nuance that smaller strides may find.
3. Padding: The pixels on the edge of an input image are “lost” as the filter collapses their values down from the kernel’s matrix to a scalar. Padding adds a dummy layer of pixels around the outside edge of the image to compensate. The smaller the input image, the more difference padding makes.

2. **Activation Functions:** After the convolution operation, an activation function, typically the Rectified Linear Unit (ReLU), is applied to introduce non-linearity into the model, making it capable of learning more complex patterns.

relu\_layer = tf.keras.layers.ReLU()

3. **Pooling Layers:** These layers reduce the spatial dimensions (width and height) of the input volume for the next convolutional layer. The most common form is max pooling, where the maximum element is selected in the region of the feature map covered by the filter.

pooling\_layer = tf.keras.layers.MaxPooling2D(pool\_size=(2, 2), strides=(2, 2))

The pooling size in a convolutional neural network determines the extent of downsampling, with larger pooling sizes reducing the spatial dimensions of feature maps more significantly, thereby compressing the input and reducing the computational load for subsequent layers.

4. **Fully Connected Layers**: At the end of a CNN, fully connected layers use the features learned during the convolutional and pooling phases to classify the input image into various classes.

# Assuming 10 classes

fc\_layer = tf.keras.layers.Dense(units=10, activation='softmax')

1. **Dense layers**: A dense layer in a neural network is a fully connected layer where each input node is connected to each output node, allowing the layer to learn complex patterns in the data.
2. **Softmax:** The softmax function is used in classification tasks to convert the output layer's raw prediction scores into probabilities, *ensuring they sum to one* and thus indicating the likelihood of each class being the correct classification.

## How CNNs Work

**Convolutional Operation:** A filter (or kernel) which is just a small matrix of weights, slides over the input image's width and height, computing the dot product between the filter and the input at each position. This operation captures the local dependencies in the input image.

**Feature Map Generation**: Each filter produces a feature map that represents the presence of specific features or patterns in the input image (vertical or horizontal lines, curves, etc.).

**ReLU Activation:** The ReLU activation function is applied to the feature map to introduce non-linear properties, allowing the network to learn complex patterns.

**Pooling**: The spatial dimensions of the feature map are reduced through pooling, which helps make the representation smaller and more manageable and provides a form of translation invariance.

**Flattening**: Before passing the output to fully connected layers, the feature map is flattened from a 2d matrix into a 1d vector.

**Classification (Fully Connected Layers):** The fully connected layers use the features to classify the input image into various classes. These layers act as a classifier on top of the features extracted by the convolutional layers and downsampled by the pooling layers.

## Example Python Code:

Here's a simple example of defining a basic CNN using TensorFlow's Keras API:

from tensorflow.keras import layers, models

model = models.Sequential()

# Convolutional layer

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

# Pooling layer

model.add(layers.MaxPooling2D((2, 2)))

# Another convolutional layer

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Another pooling layer

model.add(layers.MaxPooling2D((2, 2)))

# Another convolutional layer

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output

model.add(layers.Flatten())

# Fully connected layer

model.add(layers.Dense(64, activation='relu'))

# Output layer with 10 classes

model.add(layers.Dense(10, activation='softmax'))

model.summary() # Display the architecture of the model

In this example, the model consists of three convolutional layers, each followed by a max-pooling layer. After the convolutional and pooling layers, the feature map is flattened and fed into two fully connected layers, culminating in a **softmax** layer for classification. This architecture is common for simple image recognition tasks.

## ViTs

ViTs represent a significant shift in image processing techniques within the field of deep learning. Unlike CNNs that process images through localized filters, ViTs apply the Transformer architecture, originally designed for natural language processing, to images. Here's a detailed breakdown:

**Key Components of ViTs:**

1. **Image Tokenization:** ViTs begin by splitting an image into fixed-size patches, which are then flattened and linearly transformed into a series of vectors, known as tokens. These tokens are akin to words in a sentence for Natural Language Processing (NLP) transformers.

2. **Positional Encoding**: Since the transformer architecture does not inherently process sequential data, positional encodings are added to the tokens to retain the positional information of each patch. This step is crucial for the model to understand the arrangement of patches in the image.

3. **Transformer Encoder:** The core of a ViT is the transformer encoder, which consists of multiple layers of self-attention and feed-forward neural networks. The self-attention mechanism allows the model to weigh the importance of different patches relative to each other, enabling it to learn contextual relationships within the image.

4. **Classification Head:** After processing through the transformer encoder layers, the output is passed to a classification head, typically a simple feed-forward network, to make the final prediction.

Here's a simplified Python code snippet using TensorFlow and Keras to illustrate the architecture of a basic Visual Transformer:

import tensorflow as tf

from tensorflow.keras.layers import Dense, Dropout, LayerNormalization

from transformers import ViTFeatureExtractor, TFAutoModel

def create\_vit\_classifier(num\_classes, image\_size, patch\_size, transformer\_layers, num\_heads, mlp\_dim):

inputs = tf.keras.Input(shape=(image\_size, image\_size, 3))

# Initializing ViT feature extractor

feature\_extractor = ViTFeatureExtractor(image\_size=image\_size, patch\_size=patch\_size)

# Transforming input images to patch embeddings

patch\_embeddings = feature\_extractor(inputs)

# Transformer Encoder

for \_ in range(transformer\_layers):

x = LayerNormalization(epsilon=1e-6)(patch\_embeddings)

x = tf.keras.layers.MultiHeadAttention(num\_heads=num\_heads, key\_dim=mlp\_dim)(x, x)

x = Dropout(0.1)(x)

x = patch\_embeddings + x # Skip Connection

y = LayerNormalization(epsilon=1e-6)(x)

y = Dense(mlp\_dim, activation=tf.nn.gelu)(y)

y = Dropout(0.1)(y)

y = Dense(patch\_embeddings.shape[-1])(y)

patch\_embeddings = x + y # Skip Connection

# Classification head

# Global Average Pooling

representation = tf.reduce\_mean(patch\_embeddings, axis=1)

representation = Dropout(0.5)(representation)

outputs = Dense(num\_classes, activation='softmax')(representation)

model = tf.keras.Model(inputs=inputs, outputs=outputs)

return model

# Example usage

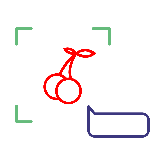
model = create\_vit\_classifier(num\_classes=10, image\_size=224, patch\_size=16, transformer\_layers=12, num\_heads=8, mlp\_dim=2048)

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

This code provides a general structure for a ViT model in TensorFlow and demonstrates the sequence of operations from image tokenization to the final classification layer. In practice, the architecture can be further customized and scaled based on the specific requirements of the task at hand.

**A red cherry with a green border

Description automatically generatedA logo of a fruit

Description automatically generated**

### Example Computer Visions Tasks

Here are some examples of the major subdisciplines in computer vision:

**Image Classification:** This is the process of categorizing images into predefined classes. Technically, it involves the use of algorithms like CNNs to process pixel data and extract features, which are then used to classify the image. It's a single-label problem when each image is assigned to one label from a set of categories (such as ImageNet’s 1,000 categories of various images).

**Object Detection:** This goes beyond classification to identify specific instances of objects within an image and their boundaries, typically represented by bounding boxes. Techniques like R-CNN (Region-based CNN) or YOLO (You Only Look Once) are employed, which involve both classifying the type of objects present and localizing them within the image.

**Object Segmentation:** This involves classifying each pixel of an image into a set of categories, resulting in a pixel-wise mask for each object.

**Semantic Segmentation:** All pixels that belong to the same object class are assigned the same label, but individual objects aren't distinguished. This means if there are three cars in an image, all cars will have the same label without differentiating between them.

**Instance Segmentation:** Unlike semantic segmentation, this not only labels pixels belonging to the same class similarly but also distinguishes between different instances of the same class. If there are three cars, each car would be uniquely identified.

**Facial Recognition:** This technology identifies or verifies a person's identity using their face. It encompasses several processes, including face detection (locating a face within an image), feature extraction (using landmarks or key points on the face), and matching against a database of known faces.

**Depth Estimation:** This technique predicts the distance from the viewpoint to the objects in a scene for each pixel. It can be achieved through methods like stereo vision, where the disparity between images from two cameras is used, or by using deep learning models trained on labeled depth data to predict depth from a single image.

**Pose Estimation:** This is about identifying the position and orientation of one or several individuals. In technical terms, pose estimation models, often deep learning-based, predict the spatial locations of key body joints (like elbows, wrists, ankles, etc.). This can be 2D, where the model predicts points on the image plane, or 3D, which involves predicting the position of points in three-dimensional space.

## Image Recognition Exercise

Now that we've thoroughly explored the basics of computer vision, let's get our hands dirty with creating an image processing model ourselves. The following exercise will give you an introduction to building your own model in code!

The notebooks for the Deep Learning Foundations course are located at [[[INSERT NOTEBOOK LINK HERE]]]