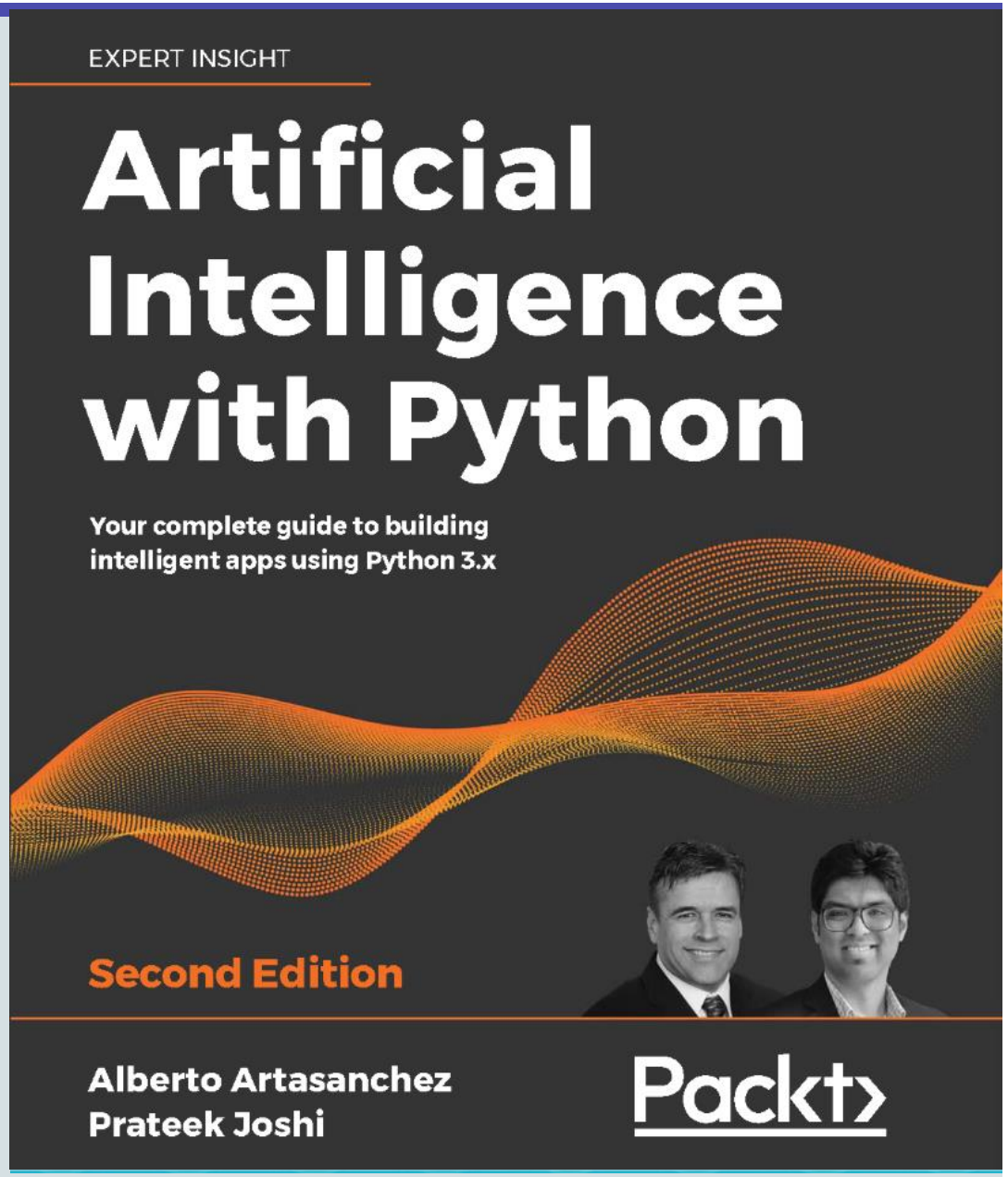
University of South-Eastern Norway
School of Business

Lecture 7 & 8: Computer Vision Artificial Neural Networks

Sinuo Wu

Course: AI for Business Applications (AI3000)



University of South-Eastern Norway
School of Business

University of South-Eastern Norway
School of Business

Do it before we start:

Download Data From Canvas – AI3000 - Files – Day7 Practice

Sinuo Wu

Course: AI for Business Applications (AI3000)

University of South-Eastern Norway
School of Business

Exam Info

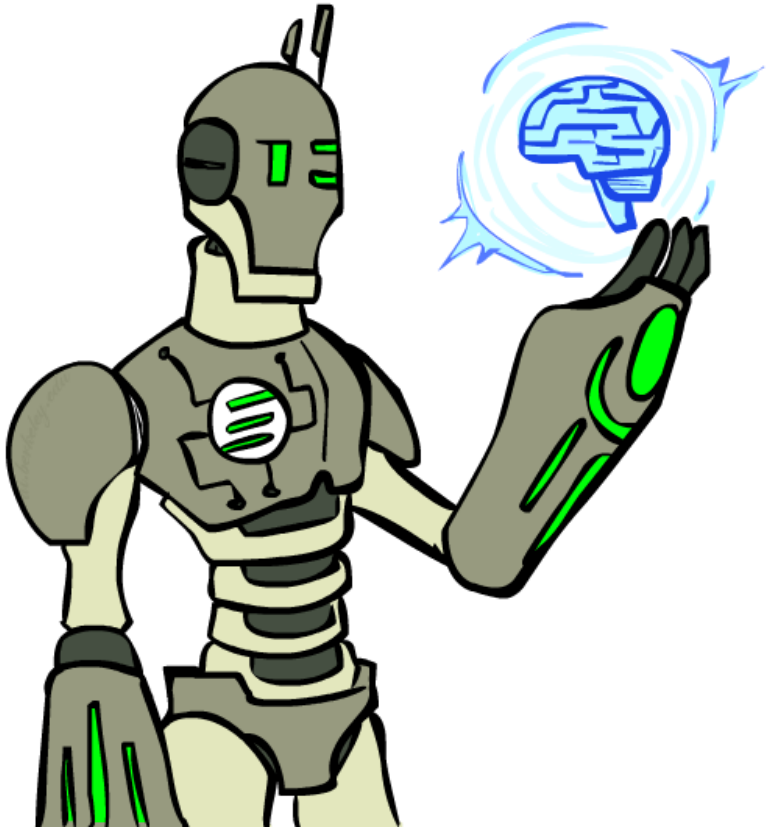
Innlev. dato	FLOW- type	Emnekode	Emne	Type	Timer/ Utlev	Rom	Ant stud på campus
Onsdag 20/11	FLOWlock	AI3000R-1	Artificial Intelligence for Business Applications	S	4 timer	**?**	58

Quiz

- Enter room number or Scan the QR code

Today

- Computer Vision
- Image Processing
- Image Feature Extraction
- Neural Networks
- Case Study



Computer Vision

Computer vision

- Computer vision is a multidisciplinary field that enables computers to interpret and understand the visual world through digital images or videos.
- Its primary goal is to replicate the human visual system's ability to extract meaningful information from visual data.
- Computer vision aims to teach machines to "see" and comprehend the visual content of images and videos.

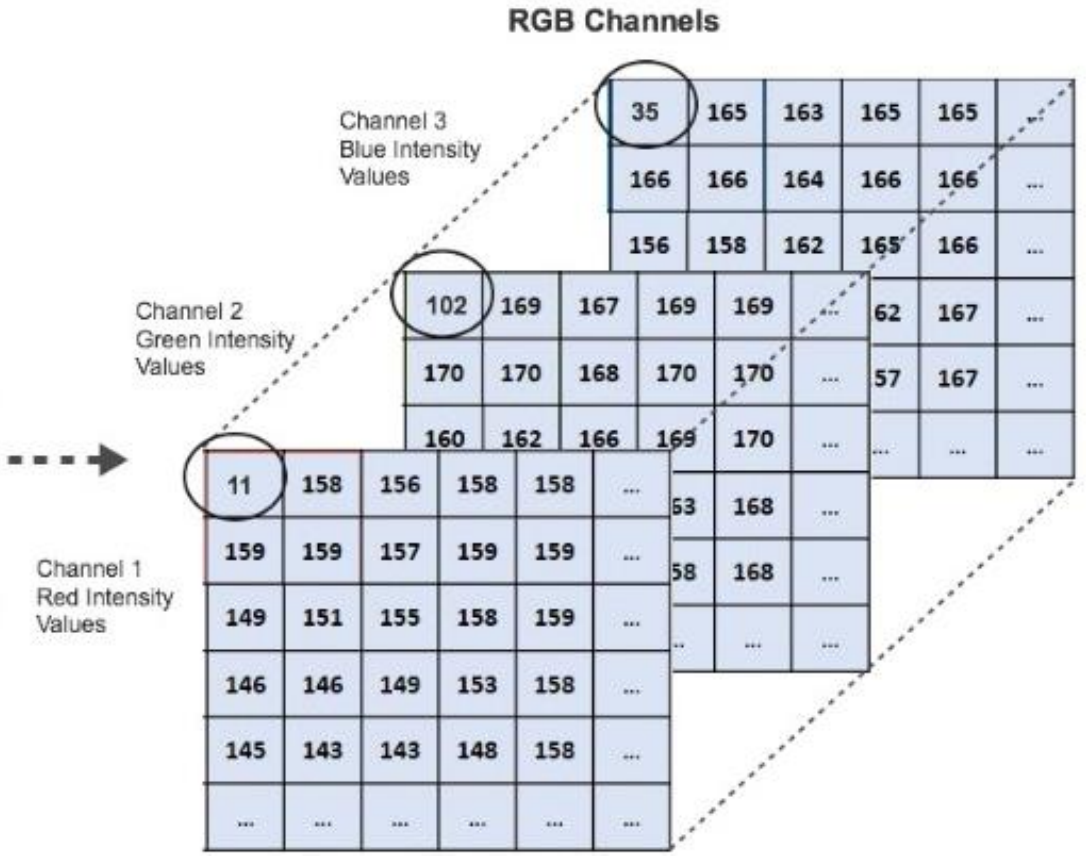
Computer vision

- **Image Processing:** The manipulation and enhancement of images to improve their quality or extract useful information.
- **Feature Extraction:** Identifying and extracting relevant patterns, shapes, or objects from images.
- **Object Detection and Recognition:** Locating and classifying objects or specific features within images or video frames.
- **Image Segmentation:** Dividing an image into meaningful regions or segments based on shared characteristics.
- **Motion Analysis:** Analyzing the movement and changes in visual content over time, often used in video processing.

Every image tells a story

- Goal of computer vision: perceive the “story” behind the picture.
- Compute properties of the world
  - 3D shape
  - Color
  - Names of people or objects
  - What happened?

Human eyes VS Computer



Challenges

Viewpoint variation

Scale

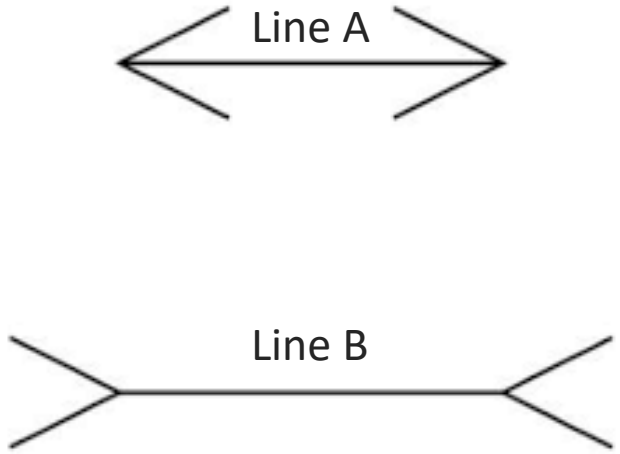
Shape

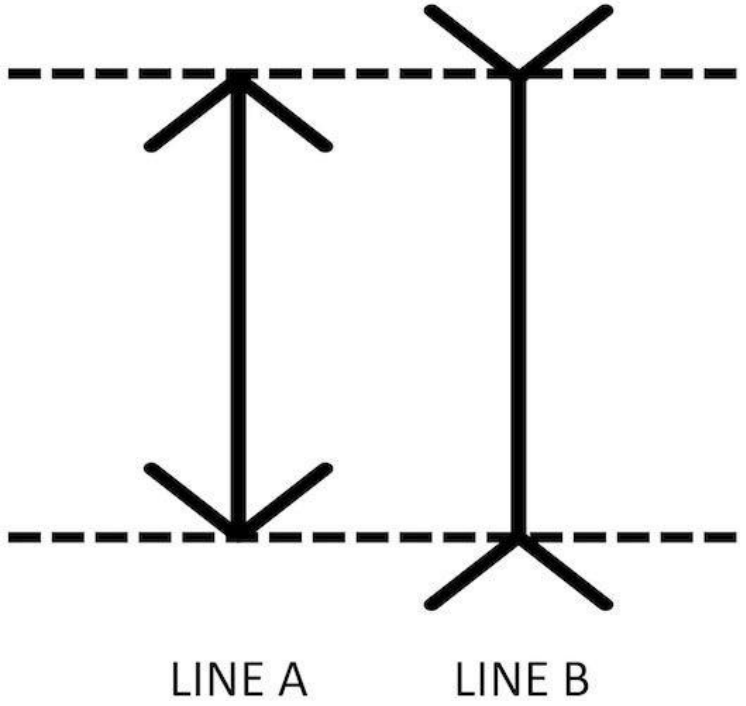
illumination

Ambiguity

Interference

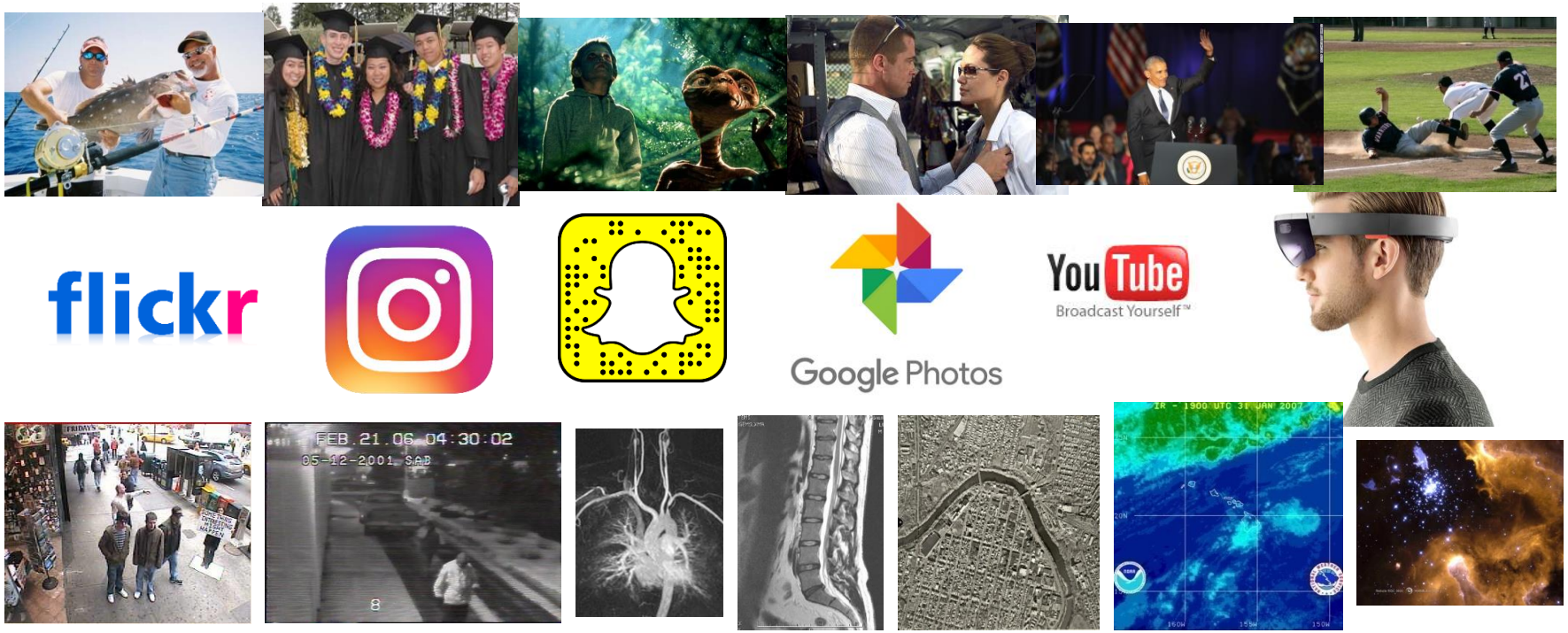
Challenges





Why Computer Vision?

- Billions of images/videos captured per day
- Huge number of applications

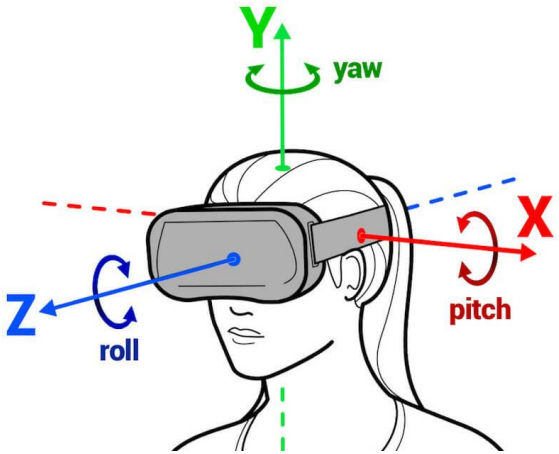


Object Detection

Waymo

EasyMile

Object Tracking



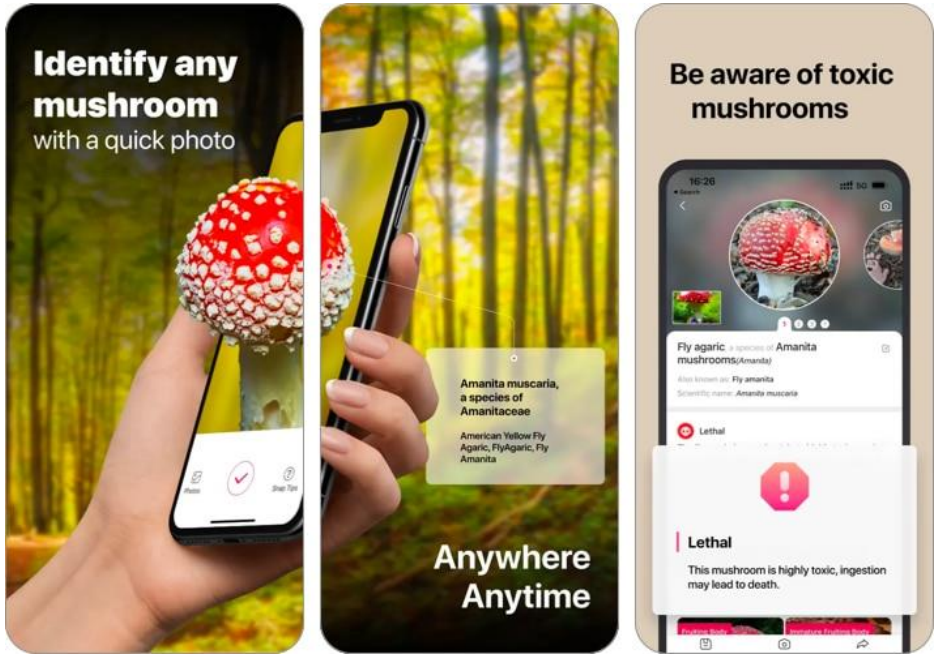
6DoF head tracking

Hand & body tracking

3D-360 video capture

Object Identification

Stargazing



Mushroom identification

Evolution

- **Early Days (1960s-1970s):** Back then, people started making computer programs to work with pictures. But these early programs couldn't handle complicated images very well.
- **Machine Learning (1980s-1990s):** Machine learning techniques began to be applied. This led to the development of algorithms for tasks such as **object recognition and image classification**.
- **Deep Learning (2000s):** In the 2000s, deep learning techniques emerged, which use neural networks to learn patterns in images and videos. These algorithms have significantly improved **the accuracy of computer vision** tasks such as object detection and facial recognition.
- **Real-Time (Recent Years):** Today, computers can do all of this in real-time. This means they can do it super fast, which is useful for things like self-driving cars, drones, and robots.

Inspiration

https://www.youtube.com/watch?v=S951cdansBI

Have a Break!

Image Processing

Image processing

- **Noise Reduction:** Images often contain noise, which is unwanted random variations in pixel values. Noise reduction techniques, such as filtering, are used to improve image quality. *Does not always mean clearer!
- **Image Enhancement:** Enhancement techniques can be applied to improve the visibility of certain features in an image, such as contrast stretching or histogram equalization.
- **Image Scaling and Resizing:** Resizing images can be done to fit them into specific dimensions or reduce file size. Interpolation methods are used to resample pixel values when resizing.

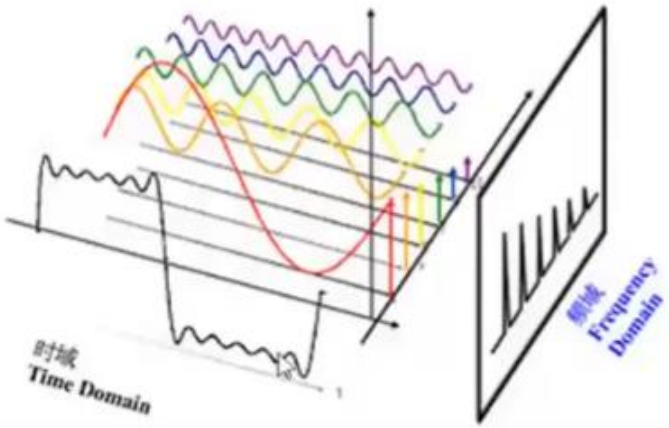
Library

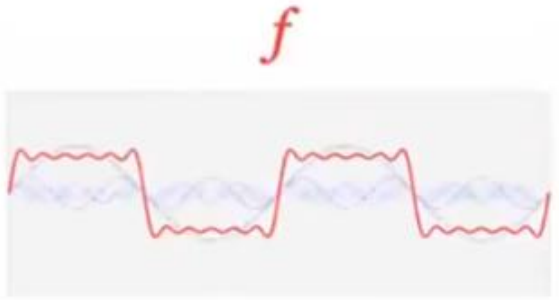
- OpenCV (OpenSource Computer Vision Library) is a versatile, open-source software library for computer vision and machine learning tasks.
- OpenCV is used in diverse applications, including image and video analysis, object detection, feature extraction, 3D vision, augmented reality, facial recognition, and more.
- Offers a rich set of functions and algorithms for image and video processing, enhancing tasks like filtering, resizing, and enhancement.
- Pip install opencv-python

Mathematical basis

- Gaussian Blur and Fourier Transform

$$A\sin(\omega x) + B\cos(\omega x)$$





$$a_n\cos(nx) + b_n\sin(nx)$$

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots\cdots$$

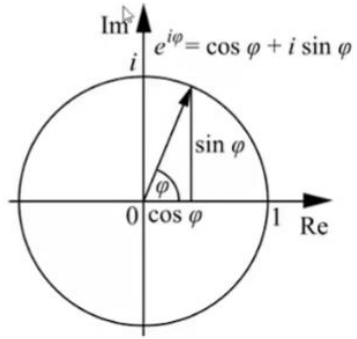
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}\cdots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}\cdots$$

欧拉公式

$$e^{ix} = \cos x + i\sin x$$

$$e^{\pi i} + 1 = 0$$



https://www.amazon.com/Machine-Learning-Algorithms-reference-algorithms/dp/1785889621

Noise Reduction

- **Try it**

```
import cv2
import numpy as np

# Load the noisy image
noisy_image = cv2.imread('noisy_image.jpg')

# Apply Gaussian blur for noise reduction
kernel_size = (5, 5)
blurred_image = cv2.GaussianBlur(noisy_image, kernel_size, 0)

# Save the denoised image
cv2.imwrite('denoised_image.jpg', blurred_image)

# Display the original and denoised images
cv2.imshow('Original Image', noisy_image)
cv2.imshow('Denoised Image', blurred_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Note: Karnel is the dimensions of a small matrix (also called a kernel or filter) that is used to perform various operations, such as blurring, sharpening, or edge detection, on an image.

Image Feature Extraction

Feature Extraction

- Feature extraction is a critical step in computer vision, where the goal is to identify relevant and informative characteristics within an image.
- These features serve as a basis for further analysis, such as object detection, image classification, or image segmentation.
- Feature extraction methods can be broadly categorized into handcrafted feature extraction and learned feature extraction using deep learning techniques.

Features in Image

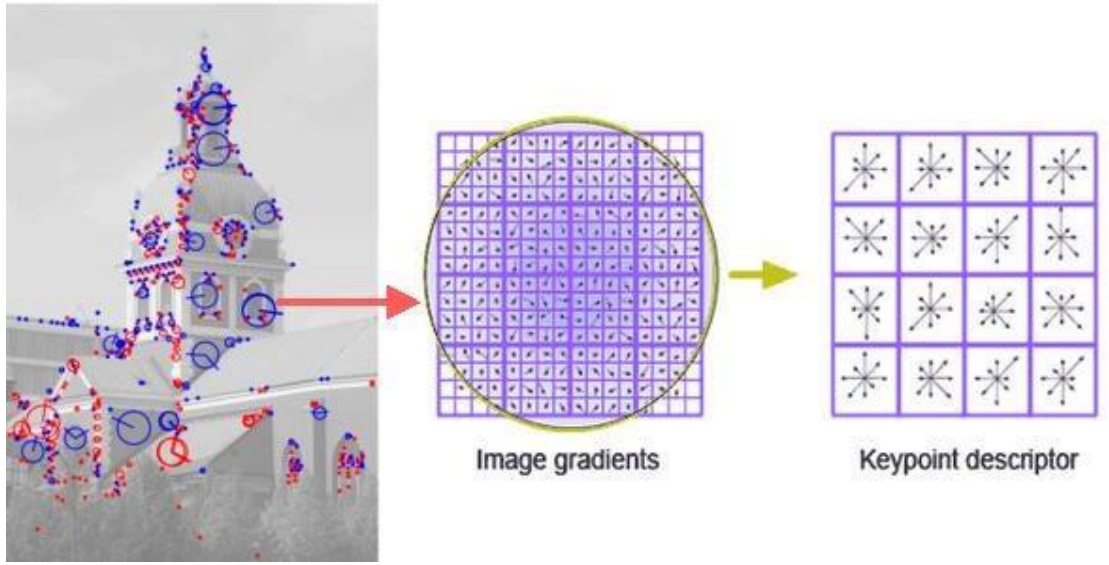
High-level features

- Color features
- Geometric features: Edge, Corner, Blob
- Texture features

Low-level features

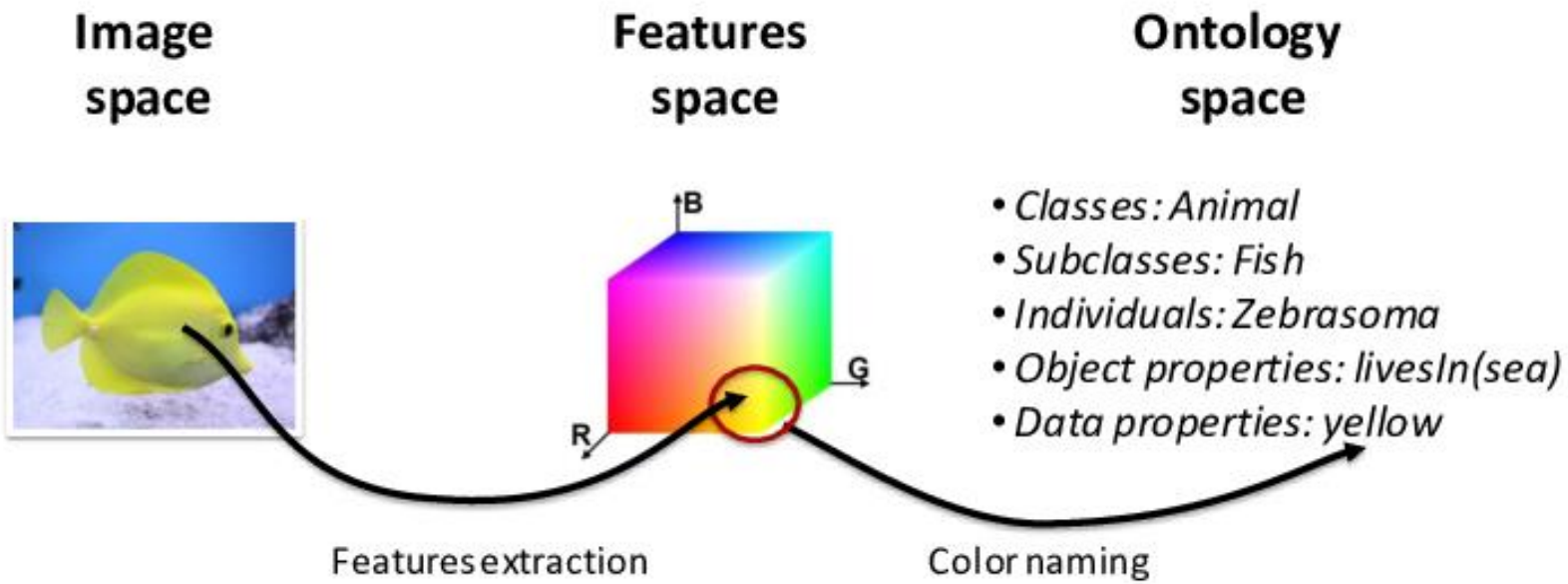
- Pixels

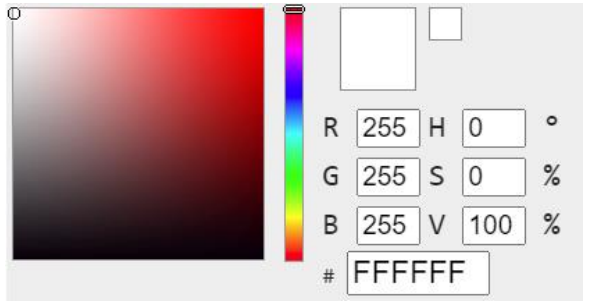
For example, **edges** involve changes in pixel intensities over a region, and **textures** look at how pixel values are spatially organized.



https://medium.com/machine-learning-world/feature-extraction-and-similar-image-search-with-opencv-for-newbies-3c59796bf7704

Color Features





Conigliaro, D., Ferrario, R., Hudelot, C., & Porello, D. (2017). Integrating Computer Vision Algorithms and Ontologies for Spectator Crowd Behavior Analysis. In *Group and Crowd Behavior for Computer Vision*. Elsevier. https://doi.org/10.1016/B978-0-12-809276-7.00016-3

Geometric Features

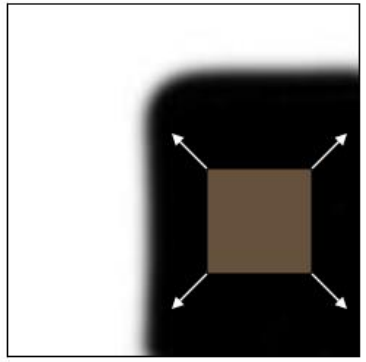
1.Edge: Represents abrupt changes in intensity in an image, often seen as lines or curves. Used for object detection and shape analysis.

2.Corner: A point where intensity changes occur in multiple directions, making them good landmarks for tasks like feature matching and image stitching.

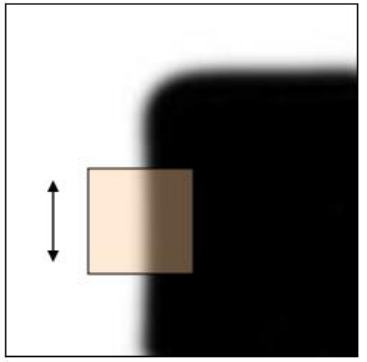
3.Blob: A region with similar intensity in an image, used for tasks like image segmentation and texture analysis.

https://cs.nyu.edu/~fergus/teaching/vision_2012/3_Corners_Blobs_Descriptors.pdf

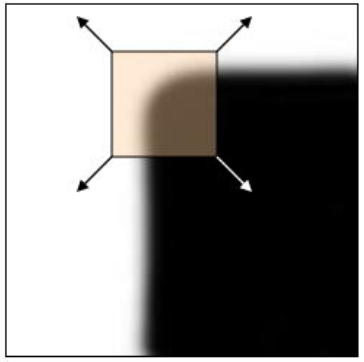
In computer vision, the detection and analysis of these features play a crucial role in **extracting meaningful information from images.** Various algorithms and techniques are employed to detect edges, corners, and blobs, and these features can be used as building blocks for more complex image processing and analysis tasks. Depending on the specific application, one or more of these features may be more suitable for capturing the desired information within an image



"Flat" region:
No change in all directions

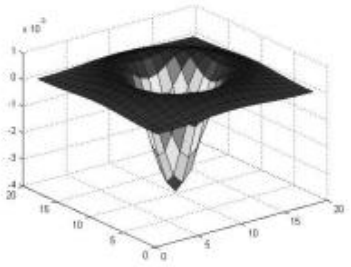


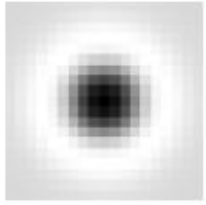
"Edge" region:
No change along edge direction



"Corner" region:
Significant change in all directions

Blob

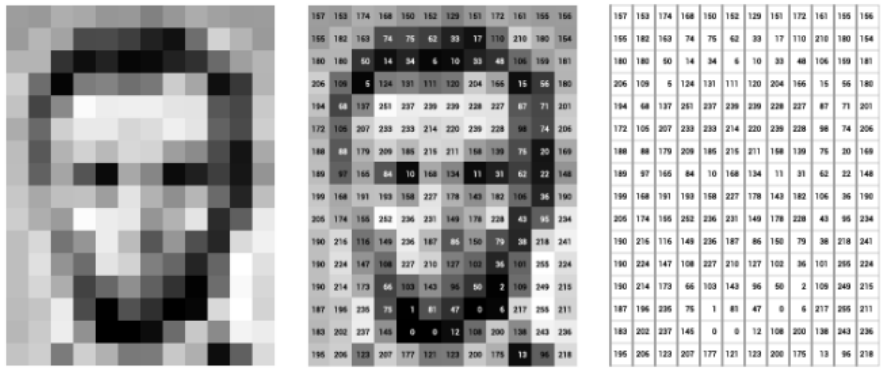


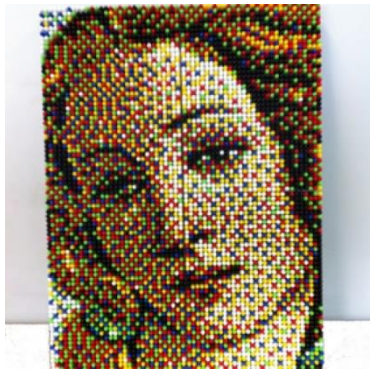


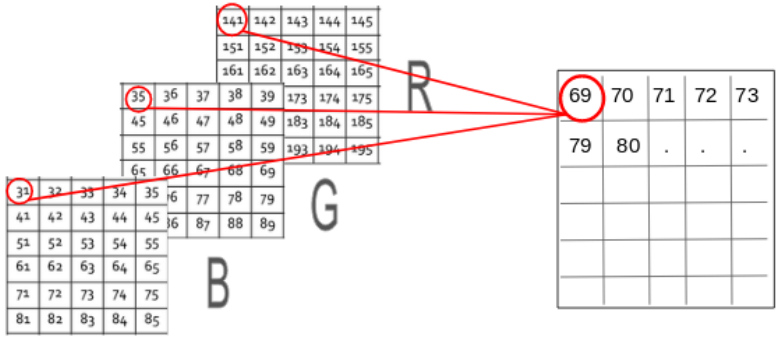
Edge

Pixel Features

- Pixel features refer to characteristics or attributes associated with individual pixels in an image.
- Pixel features are typically represented as **numerical values**.
- Pixel features are the result **of image transformation**, this process can be achieved by algorithms.
- Each pixel in an image is associated with a set of numerical attributes or characteristics that describe various properties of that pixel, such as its color intensity, position, and so on.
- For example, in grayscale images, the pixel's intensity is represented by a single numerical value ranging from 0 (black) to 255 (white).







Algorithms

1.SIFT (Scale-Invariant Feature Transform):

- **Description:** SIFT is an algorithm that extracts key points and their descriptors from an image. It is known for its scale and rotation invariance, making it robust to changes in an object's size and orientation.
- **Key Points:** SIFT detects distinctive local features, such as corners and blobs, and computes descriptors that capture information about the surrounding region's texture and gradient.

2.Frame Differencing

- **Description:** Frame differencing is an algorithm used for motion detection in video streams. It analyzes consecutive frames to identify areas of significant change, indicating movement.
- **Key Points:** Frame differencing works by calculating the absolute differences between successive frames. By comparing the current frame with the previous and next frames, it highlights regions where motion occurs. This method is effective for detecting simple movements but may struggle in scenarios with changing lighting conditions or camera shake.

3.CAMShift (Continuously Adaptive Mean Shift):

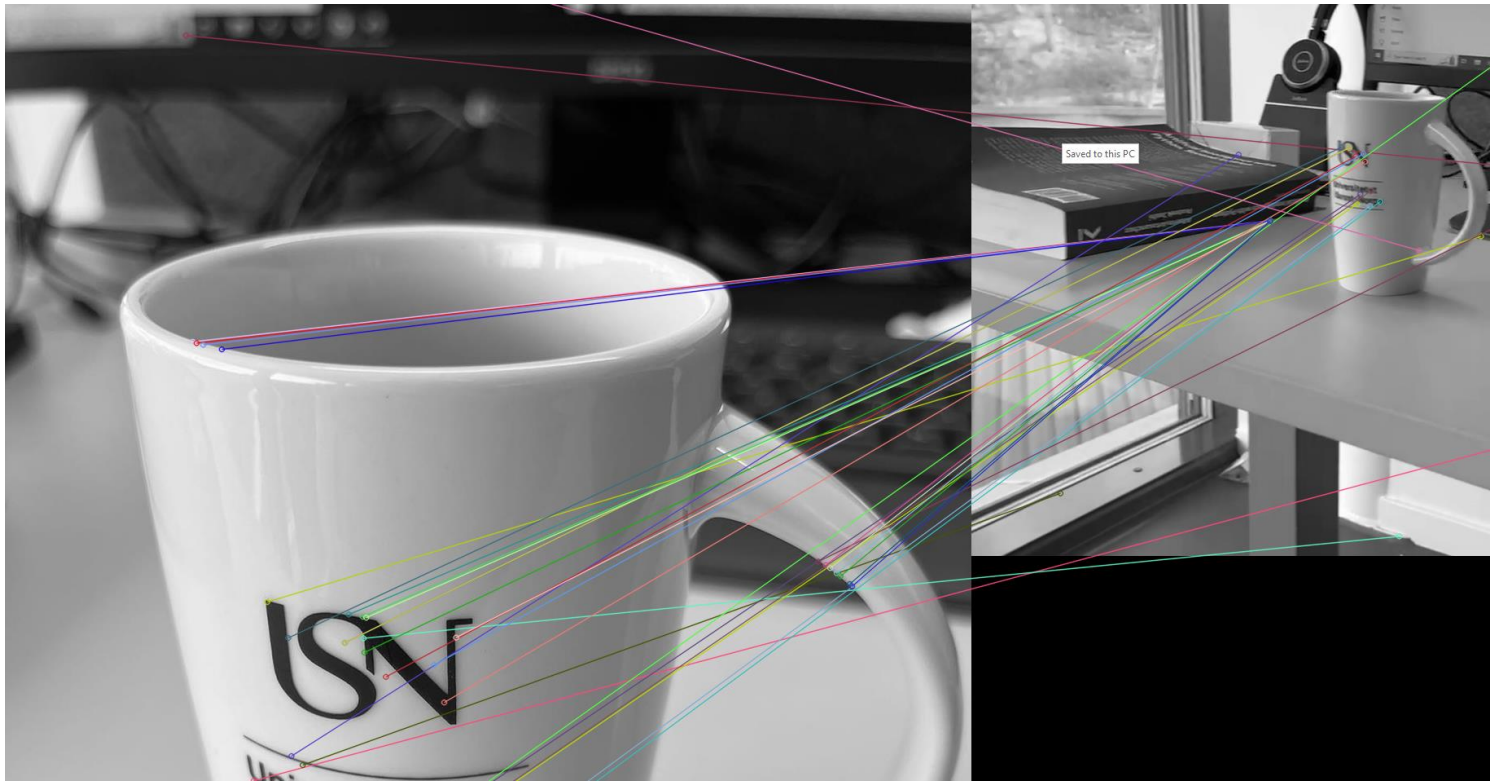
- **Description:** CAMShift is an algorithm used for object tracking in video streams. It adapts the mean shift algorithm to continuously update the position and size of the target object by analyzing its color histogram.
- **Key Points:** CAMShift tracks objects by using their color distribution and adjusts the search window size based on the object's dimensions, making it effective for objects that change in size and shape.

Case Study

SIFT

- **Load the Images, then create SIFT Detector**.
- **Detect Keypoints and Compute Descriptors**: This step generates a set of key points (such as edges, corners, or blobs), and then compute the **descriptor (feature vectors)** for each key point that represents its appearance.
- **Create a Matcher**: Initialize a feature matcher. Common choices include FLANN (Fast Library for Approximate Nearest Neighbors) or BFMatcher (Brute Force Matcher).
- **Match Keypoints**: Match the descriptors between the reference image and the target image using the matcher. If using FLANN, use the kNNMatch method to get the k-nearest matches.
- **Filter Good Matches**: Apply a ratio test (e.g., Lowe's ratio test) to filter out weak matches. Keep only the matches that are significantly better than the second-best match.
- **Draw Matches**: Use cv2.drawMatches to visualize the matched key points between the reference image and the target image.
- **Display Results**: Use cv2.imshow() to display the result with drawn matches.

Case: Detect an object from a video



Simple version of object detection

Codes

```
import cv2

# Load the image and create the SIFT detector
reference_image = cv2.imread('cup.jpg', cv2.IMREAD_GRAYSCALE)
sift = cv2.SIFT_create()

# Find keypoints and descriptors in the reference image
kp1, des1 = sift.detectAndCompute(reference_image, None)

# Create a FLANN-based matcher
index_params = dict(algorithm=1, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)

# Load and Initialize a video capture object
cap = cv2.VideoCapture('test_video.mp4')

# Set up the window
cv2.namedWindow('Object Detection', cv2.WINDOW_NORMAL)
cv2.resizeWindow('Object Detection', 800, 600)
```

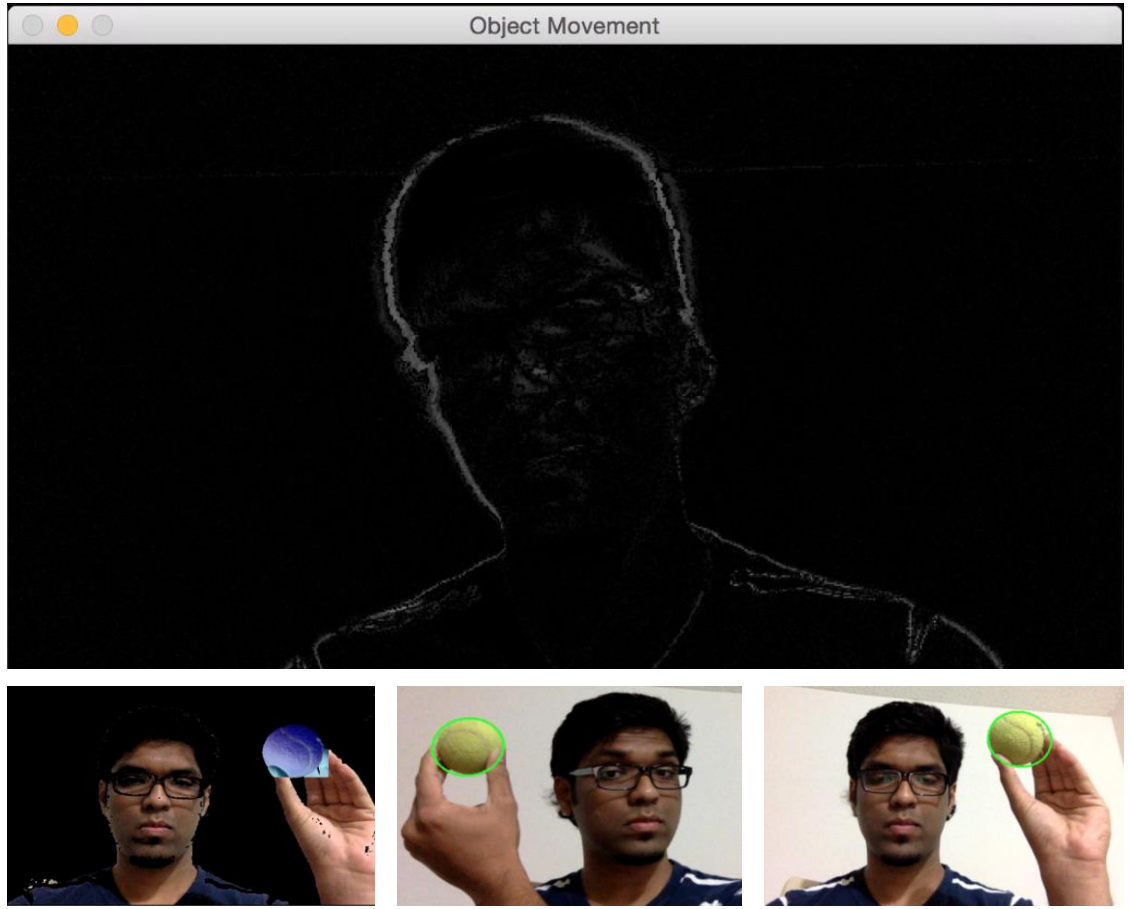
Codes

```
while True:
    # Read a frame from the video
    ret, frame = cap.read()

    if not ret:
        break

    # Convert the frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Find keypoints and descriptors in the frame
    kp2, des2 = sift.detectAndCompute(gray_frame, None)

    # Match keypoints between the reference image and the frame
    matches = flann.knnMatch(des1, des2, k=2)

    # Filter good matches. Apply ratio test to get good matches
    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)
```

Codes

```
    # Draw matches on the frame
    result_frame = cv2.drawMatches(reference_image, kp1, gray_frame, kp2, good_matches, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

    # Resize the result_frame for better visibility
    result_frame = cv2.resize(result_frame, (800, 600))  # Resize to a fixed size

    # Display the resulting frame
    cv2.imshow('Object Detection', result_frame)

    # Exit the loop if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release video capture and close all windows
cap.release()
cv2.destroyAllWindows()
```

Object tracking




```
import cv2

# Compute the frame differences
def frame_diff(prev_frame, cur_frame, next_frame):
    # Difference between the current frame and the next frame
    diff_frames_1 = cv2.absdiff(next_frame, cur_frame)

    # Difference between the current frame and the previous frame
    diff_frames_2 = cv2.absdiff(cur_frame, prev_frame)

    return cv2.bitwise_and(diff_frames_1, diff_frames_2)

# Define a function to get the current frame from the webcam
def get_frame(cap, scaling_factor):
    # Read the current frame from the video capture object
    _, frame = cap.read()

    # Resize the image
    frame = cv2.resize(frame, None, fx=scaling_factor,
            fy=scaling_factor, interpolation=cv2.INTER_AREA)

    # Convert to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    return gray

if __name__=='__main__':
    # Define the video capture object
    cap = cv2.VideoCapture(0)

    # Define the scaling factor for the images
    scaling_factor = 0.5

    # Grab the current frame
    prev_frame = get_frame(cap, scaling_factor)

    # Grab the next frame
    cur_frame = get_frame(cap, scaling_factor)

    # Grab the frame after that
    next_frame = get_frame(cap, scaling_factor)

    # Keep reading the frames from the webcam
    # until the user hits the 'Esc' key
    while True:
        cv2.imshow('Object Movement', frame_diff(prev_frame,
                cur_frame, next_frame))

        prev_frame = cur_frame
        cur_frame = next_frame
        next_frame = get_frame(cap, scaling_factor)

        key = cv2.waitKey(10)
        if key == 27:
            break

    # Close all the windows
    cv2.destroyAllWindows()
```

Frame Differ

CAMShift Algorithm

1. **Initialization**: Begin by selecting an initial region of interest (ROI) around the object you want to track. This region is typically defined using a bounding box.
2. **Histogram Creation**: Create a color histogram based on the color information within the selected ROI. This histogram represents the object's color distribution.
3. **Back projection**: Calculate a back projection of the histogram. In this step, each pixel in the entire image is assigned a value indicating how closely it matches the colors in the histogram. This helps locate the object in subsequent frames.
4. **Mean Shift Tracking**: Apply the Mean Shift algorithm to iteratively find the peak (mode) of the back projection. This involves shifting the region towards the peak's location based on the **similarity of pixel colors to the histogram**.
5. **Adaptive Sizing and Orientation**: CAMShift continuously adapts the size and orientation of the tracking window based on the object's characteristics. It computes an oriented bounding box to enclose the object, considering changes in scale and orientation.
6. **Repeating Tracking**: Repeatedly perform the Mean Shift tracking and adaptive sizing and orientation steps for each frame in a video sequence to continually update the object's position, size, and orientation.
7. **Termination**: The tracking process continues until the end of the video sequence or until the user decides to stop it.

CAMShift

```python
import cv2
import numpy as np

# Define a class to handle object tracking related functionality
class ObjectTracker(object):
    def __init__(self, scaling_factor=0.5):
        # Initialize the video capture object
        self.cap = cv2.VideoCapture(0)

        # Capture the frame from the webcam
        _, self.frame = self.cap.read()

        # Scaling factor for the captured frame
        self.scaling_factor = scaling_factor

        # Resize the frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)

        # Create a window to display the frame
        cv2.namedWindow('Object Tracker')

        # Set the mouse callback function to track the mouse
        cv2.setMouseCallback('Object Tracker', self.mouse_event)

        # Initialize variable related to rectangular region selection
        self.selection = None

        # Initialize variable related to starting position
        self.drag_start = None

        # Initialize variable related to the state of tracking
        self.tracking_state = 0

    # Define a method to track the mouse events
    def mouse_event(self, event, x, y, flags, param):
        # Convert x and y coordinates into 16-bit numpy integers
        x, y = np.int16([x, y])
```

Load the data:
Real time data the video captured

```python
# Method to start tracking the object
def start_tracking(self):
    # Iterate until the user presses the Esc key
    while True:
        # Capture the frame from webcam
        _, self.frame = self.cap.read()

        # Resize the input frame
        self.frame = cv2.resize(self.frame, None,
                fx=self.scaling_factor, fy=self.scaling_factor,
                interpolation=cv2.INTER_AREA)

        # Create a copy of the frame
        vis = self.frame.copy()

        # Convert the frame to HSV colorspace
        hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)

        # Create the mask based on predefined thresholds
        mask = cv2.inRange(hsv, np.array((0., 60., 32.)),
                    np.array((180., 255., 255.)))

        # Check if the user has selected the region
        if self.selection:
            # Extract the coordinates of the selected rectangle
            x0, y0, x1, y1 = self.selection

            # Extract the tracking window
            self.track_window = (x0, y0, x1-x0, y1-y0)

            # Extract the regions of interest
            hsv_roi = hsv[y0:y1, x0:x1]
            mask_roi = mask[y0:y1, x0:x1]

            # Compute the histogram of the region of
            # interest in the HSV image using the mask
            hist = cv2.calcHist( [hsv_roi], [0], mask_roi,
                    [16], [0, 180] )
```

Standardization

It allows the user to select a region of interest (ROI) by clicking and dragging the mouse

Have a Break!

Neural Networks

Neural Network

- An Artificial Neural Network (ANN) is a computational model inspired by the structure and function of the **human brain**. It is a fundamental component of machine learning and deep learning, designed to process and learn from data to make predictions or decisions.

- ANNs consist of interconnected nodes, often referred to as neurons or units, organized into layers. These layers typically include an input layer, one or more hidden layers, and an output layer.

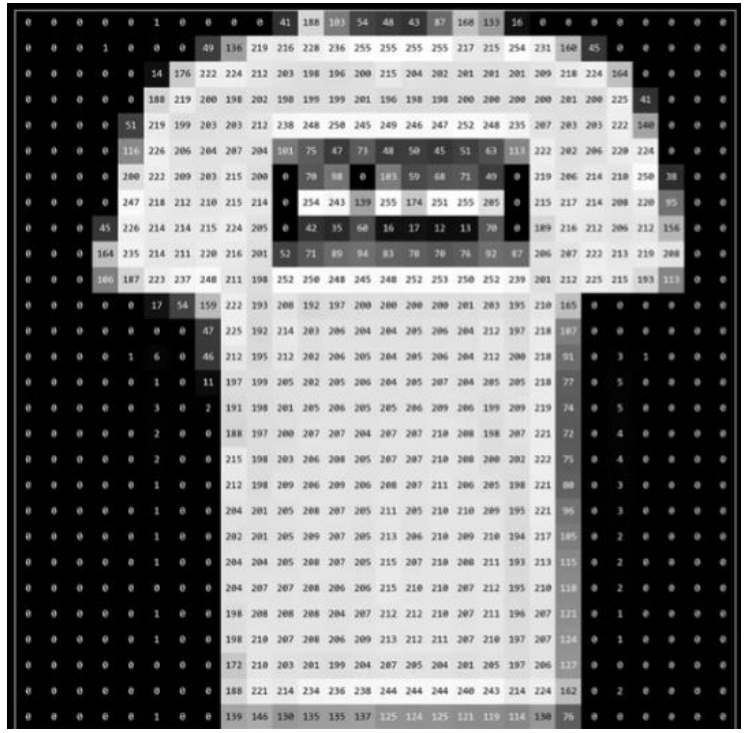
- ANNs are used in pattern recognition, regression, classification, computer vision, NLP, and deep learning, and have revolutionized machine learning.

Neural Network

- **Neurons:** Nodes that process and transmit information.
- **Layers:** Organization into input, hidden, and output layers.
- **Weights and Biases:** Adjusted during training to minimize errors.
- **Learning:** The process of adjusting weights to improve predictions.
- **Backpropagation:** Algorithm for updating weights during training.
- **Activation Functions:** Determine if neurons are activated.

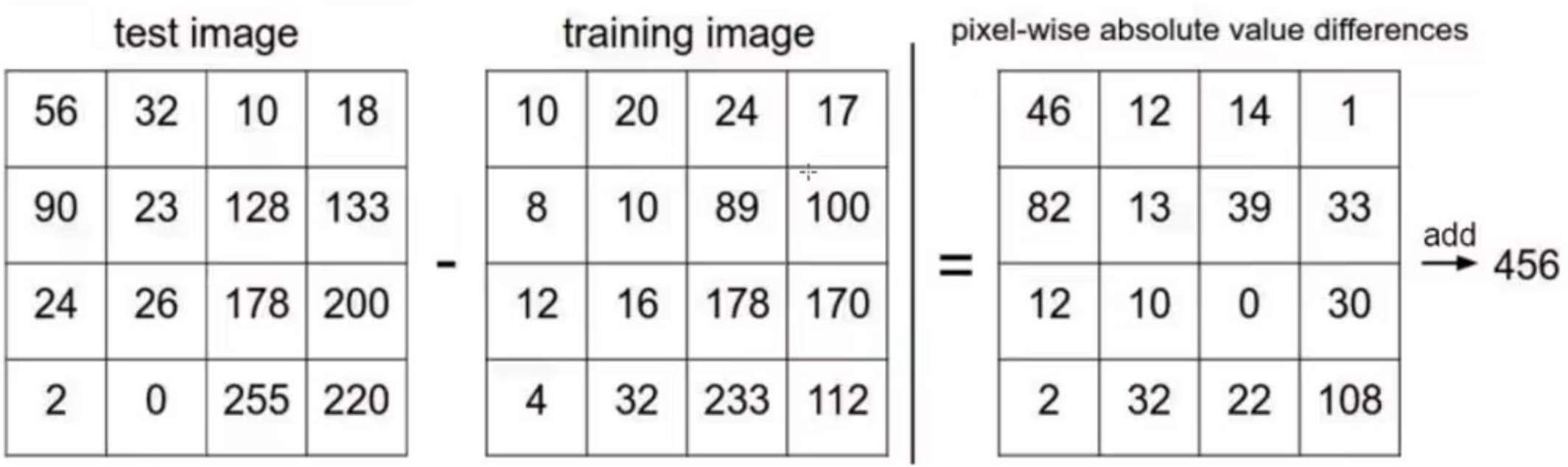
Example

Pixel Features

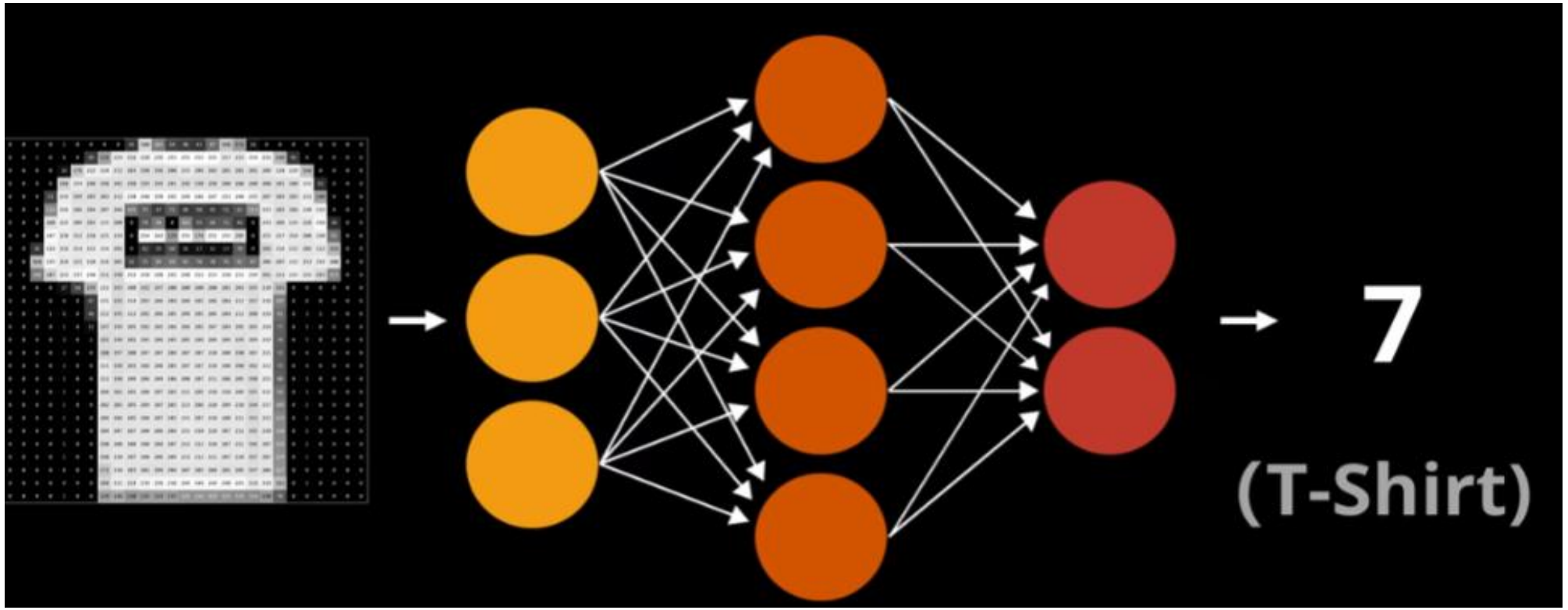


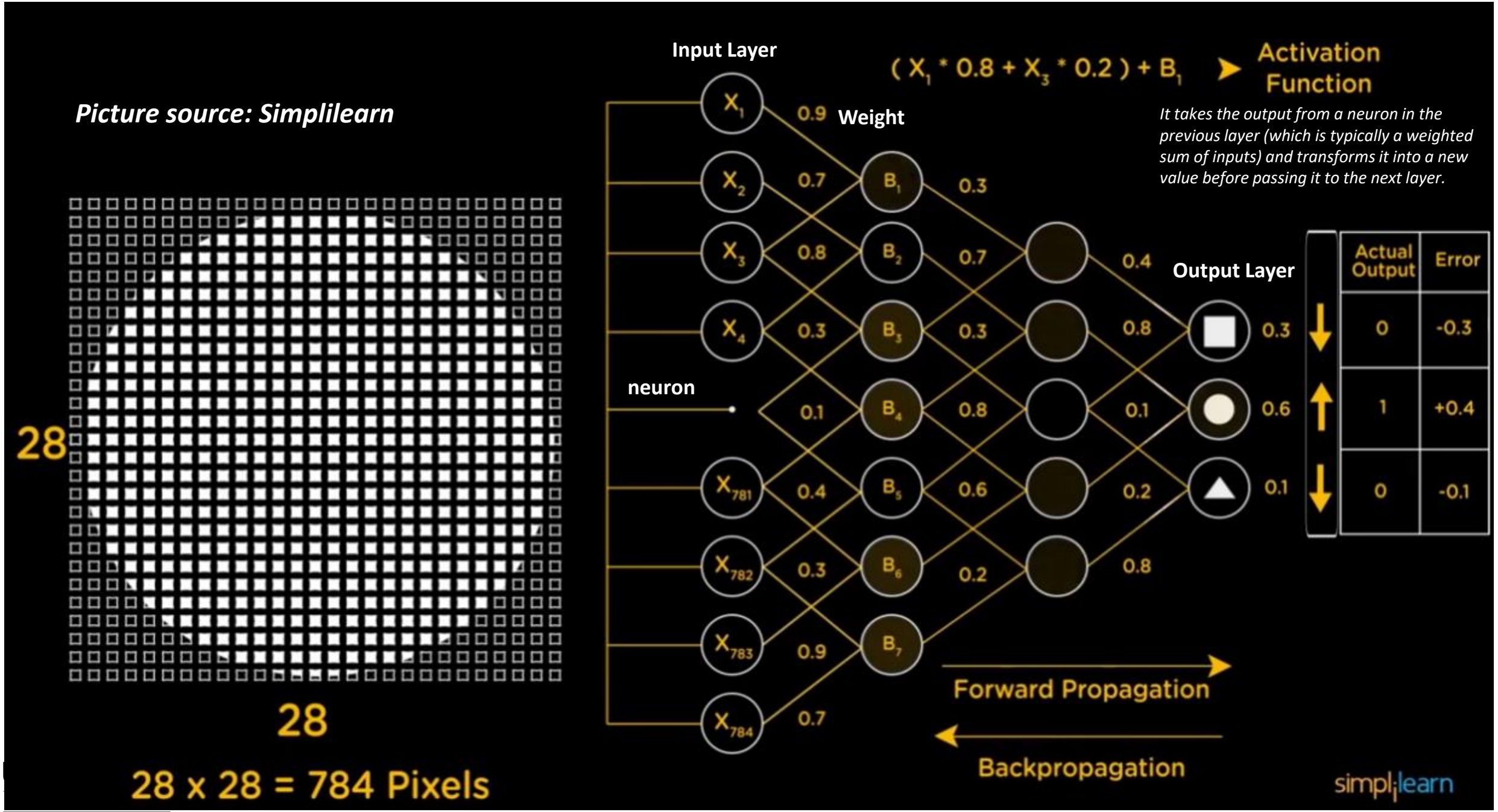
$f(x)$ → **7** → T-Shirt

Example

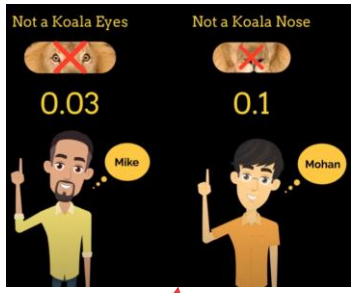

test image
training image
pixel-wise absolute value differences
56 32 10 18
90 23 128 133
24 26 178 200
2 0 255 220
-
10 20 24 17
8 10 89 100
12 16 178 170
4 32 233 112
=
46 12 14 1
82 13 39 33
12 10 0 30
2 32 22 108
add
456

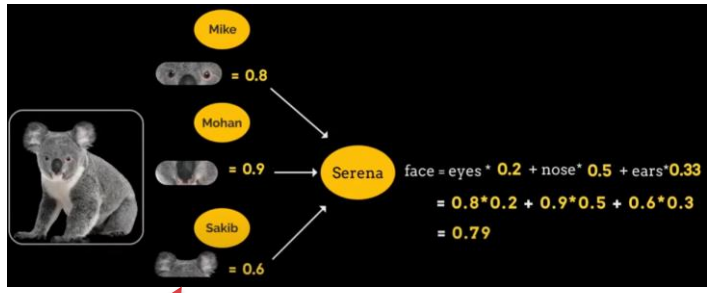
Neural Network

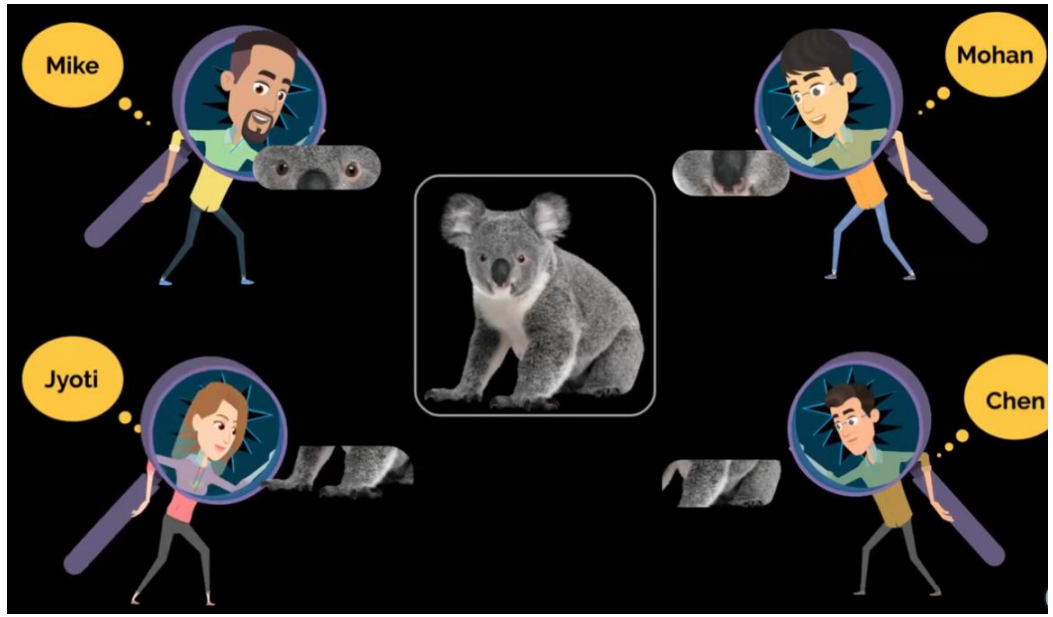


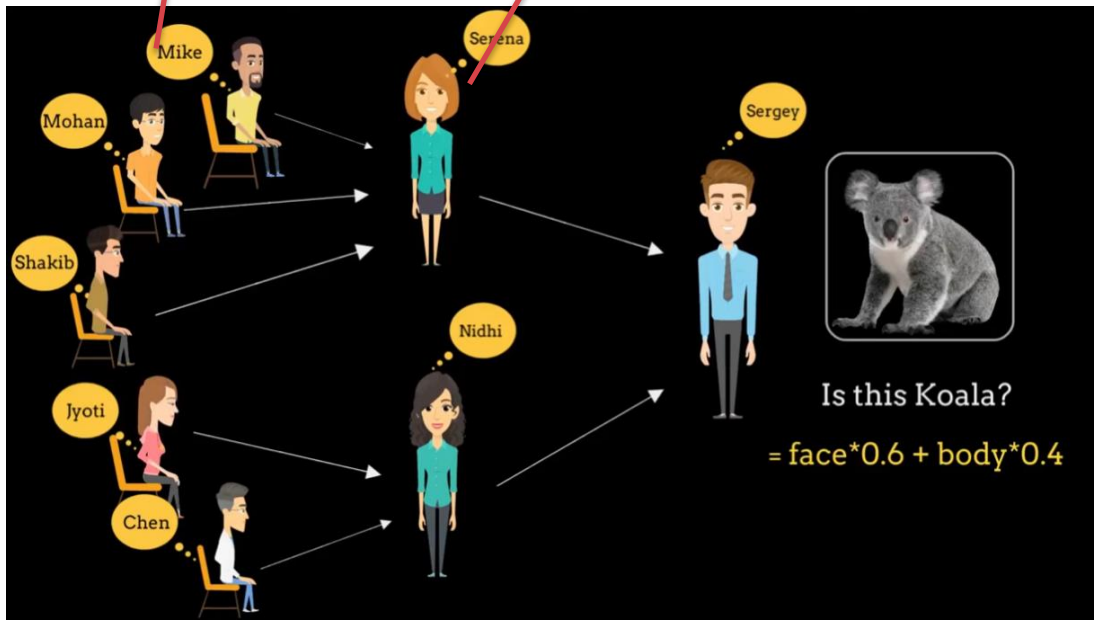

Picture source: Simplilearn
28
28
28 x 28 = 784 Pixels
Input Layer
$(X_1 * 0.8 + X_3 * 0.2) + B_1$
Activation Function
It takes the output from a neuron in the previous layer (which is typically a weighted sum of inputs) and transforms it into a new value before passing it to the next layer.
Weight
neuron
Output Layer
Actual Output
Error
0
-0.3
1
+0.4
0
-0.1
Forward Propagation
Backpropagation
simplilearn

Neural Network

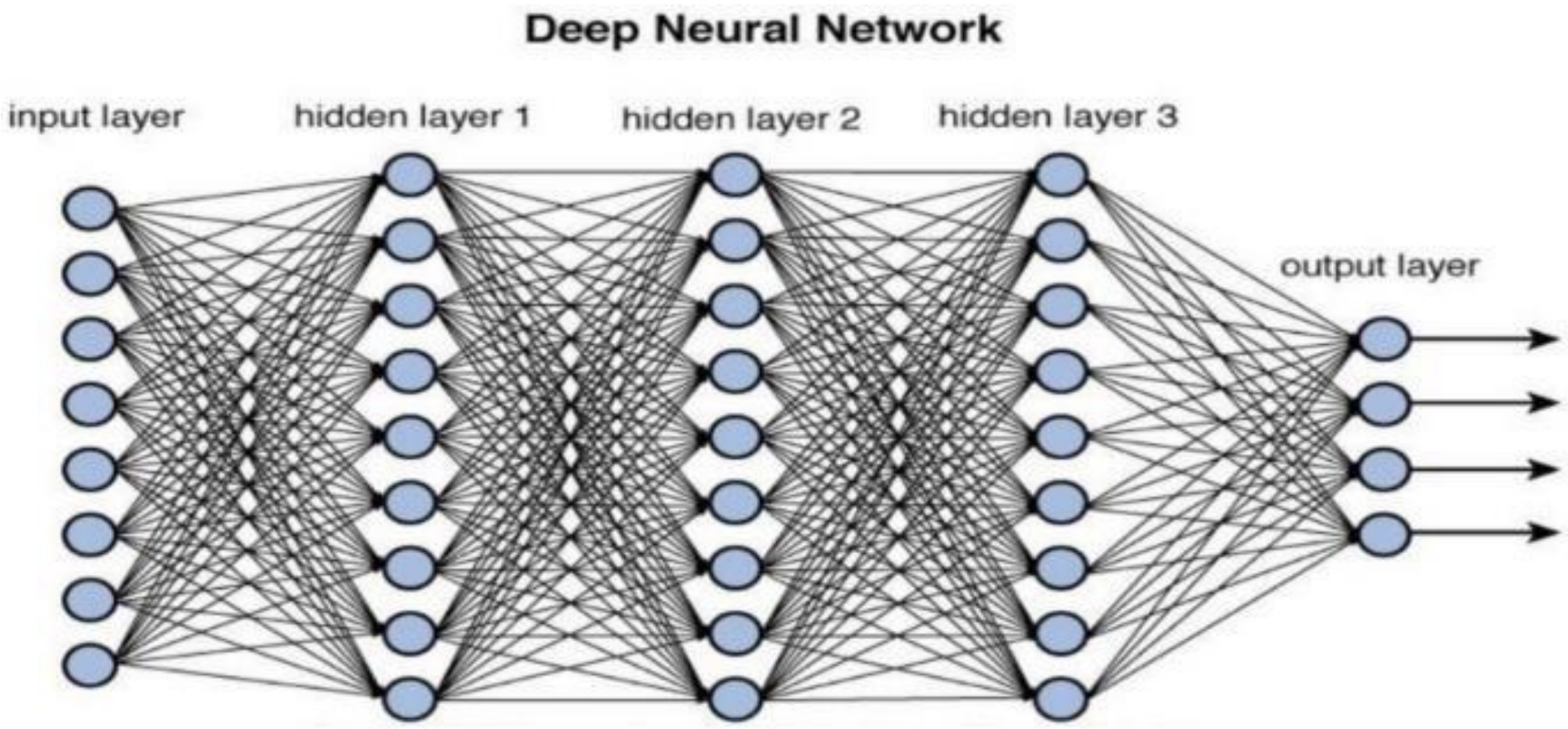








Deep Neural Network



(w * x + b)

Training a Neural Network

- If we are dealing with *N*-dimensional input data, then the input layer will consist of *N* neurons.
- If we have *M* distinct classes in our training data, then the output layer will consist of *M* neurons.
- A simple neural network will consist of a couple of layers and a deep neural network will consist of many layers.

Training a Neural Network

- So how can a neural network be used to classify data?
  - The first step is to collect the appropriate training data and label it.
  - Each neuron acts as a simple function and the neural network trains itself until the error goes below a certain a threshold.
  - The error is the difference between the predicted output and the actual output.
  - Based on how big the error is, the neural network adjusts itself and retrains until it gets closer to the solution.

When to use it

- Neural networks are universal approximators, and they work best if the system you are using them to model has a high tolerance to error.
  - Capturing associations or discovering regularities within a set of patterns;
  - Where the volume, number of variables or diversity of the data is very great;
  - The relationships between variables are vaguely understood; or,
  - The relationships are difficult to describe adequately with conventional approaches.

http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

Applications

Image and Video Analysis: Image classification, object detection, video analysis.

Natural Language Processing: Language translation, sentiment analysis, chatbots.

Speech Recognition and Synthesis: Speech-to-text, text-to-speech.

Autonomous Vehicles: Self-driving cars.

Healthcare and Medicine: Disease diagnosis, drug discovery, patient risk assessment.

Finance: Stock market prediction, credit scoring, fraud detection.

Robotics: Object manipulation, navigation.

Industrial and Manufacturing: Quality control, predictive maintenance.

Energy and Environmental Applications: Energy consumption prediction, environmental monitoring.

Anomaly Detection: Cybersecurity and fraud detection.

TensorFlow

Google's open-source machine learning Library.

- **Deep Learning:** Especially well-suited for deep neural networks.
- **Flexibility:** Uses a computational graph for customization.
- **High Performance:** Optimized for CPU and GPU, ideal for large datasets.
- **Community and Ecosystem:** Supported by a large community and rich tools.
- **Deployment**: Suitable for deploying models in various environments.
- **Research:** Used for cutting-edge machine learning research.
- **Open Source:** Freely available and actively maintained.

Case Study

Case

Train a model to Predict the type of flowers

Libraries

1. Matplotlib: This library is used for creating static, interactive, and animated visualizations in Python. It helps visualize training and validation metrics, such as accuracy and loss.

2. Numpy: NumPy is a library for numerical computations in Python, providing support for arrays and mathematical functions. It is commonly used for data manipulation and processing in machine learning.

3. Pathlib: Pathlib offers an object-oriented approach to handle filesystem paths, making it easier to manage file paths across different operating systems. It simplifies tasks like defining directories for datasets.

4. TensorFlow: It is a powerful open-source framework for building and deploying machine learning models. It provides tools for model construction, training, and data preprocessing.

5. Keras: It is API within TensorFlow that provides various neural network layers to build deep learning models. It simplifies the process of adding and configuring layers in a model architecture.

6. Sequential: The Sequential class allows for creating a linear stack of layers in a neural network. It provides a straightforward way to build models layer by layer.

Codes

```
# Manually downloaded dataset directory
FLOWERS_DIR = './flower_photos'
data_dir = pathlib.Path(FLOWERS_DIR)

# Check the number of images
image_count = len(list(data_dir.glob('*/*.jpg')))
print(f"Total images: {image_count}")

# Define parameters
batch_size = 32
img_height = 180
img_width = 180

# Load and split the dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

Iterations=total samples/batch size

For example, if your training dataset contains 1,000 images and your batch size is 32, it will take 32 iterations to complete one epoch (one full pass through the entire dataset)

```
Total images: 3670
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
Class names: ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

https://www.tensorflow.org/tutorials/images/classification

Codes

```
# Class names (flower types)
class_names = train_ds.class_names
print(f"Class names: {class_names}")

# Normalize pixel values (rescale images to [0, 1] range)
normalization_layer = layers.Rescaling(1./255)

# Build the model without data augmentation using Input layer
model = Sequential([
    layers.Input(shape=(img_height, img_width, 3)),
    normalization_layer,
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])

# Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# Train the model
epochs = 3
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs
)

# Visualize training results
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

# Save the model
model.save('flower_classifier_model.h5')
```

Apply model

```python
import tensorflow as tf
import numpy as np
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Use your model instead of the pre-defined model
model = MobileNetV2(weights='imagenet')

# Function to load and preprocess an image from a file path
def load_and_preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    return img_array

# Inference on new images
new_image_path = 'C:\\Users\\ws\\Desktop\\Test.jpg'  # Replace with the path to your new image

# Load and preprocess the new image
new_image = load_and_preprocess_image(new_image_path)

# Get predictions for the new image
predictions = model.predict(new_image)
decoded_predictions = decode_predictions(predictions, top=5)[0]

# Display the top 5 predicted classes
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print("Prediction {}: {} ({:.2f}%)".format(i + 1, label, score * 100))

# Display the image
img = image.load_img(new_image_path)
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 1s 557ms/step
Prediction 1: daisy (92.40%)
Prediction 2: cup (0.10%)
Prediction 3: fly (0.08%)
Prediction 4: bee (0.07%)
Prediction 5: snail (0.06%)
```

Have a Break!

Practice

Try to mimic the codes for:

1. Classify fire
2. Classify mushroom

You can find related data from Kaggle

THANKS