NEXUS AI DIGITAL

Computer Vision Tasks

Welcome to the Nexus AI Vision team. Your mission is to develop the software for a new smart security camera. You will teach the camera how to see, understand, and interpret the world around it, progressing from analyzing still images to processing live video feeds.

Part 1: Beginner Level (The Image Analyst)

Focus: Handling individual images and understanding their content.

Task 1: The First Frame (Image Fundamentals)

- **The "Why":** Before a camera can understand a scene, it must first be able to process a single image. This task covers the fundamental building blocks of computer vision.
- **Description:** Learn the basics of image manipulation and preprocessing using the OpenCV library.
- **Step-by-Step Guide:**
    1. **Load & Display:** Use OpenCV to load an image from a file and display it.
    2. **Image Properties:** Access basic properties like height, width, and number of channels.
    3. **Color Spaces:** Convert the image to different color spaces, especially Grayscale.
    4. **Basic Operations:** Resize the image, apply a Gaussian blur to reduce noise, and use the Canny edge detector to find edges.
- **Suggested Dataset:** A simple set of high-quality images of outdoor scenes. Intel Image Classification Dataset on Kaggle.
- **Expected Outcome:** A Jupyter Notebook demonstrating each of the image manipulation techniques on a sample image, with brief explanations for each step.

Task 2: What's in the Picture? (Image Classification)

- **The "Why":** The camera needs to identify the main subject of a scene. Is it a person, a car, or an animal? This is the first step towards an "intelligent" system.
- **Description:** Use a powerful, pre-trained deep learning model to classify the content of various images.
- **Step-by-Step Guide:**
    1. **Load a Pre-trained Model:** Use TensorFlow and Keras to load a well-known model like **MobileNetV2** or **ResNet50**, complete with its pre-trained ImageNet weights.
    2. **Image Preprocessing:** Write a function to prepare an image for the model (resizing to 224x224, normalizing pixel values).
    3. **Make Predictions:** Feed several different images (e.g., a cat, a car, a tree) into the model.
    4. **Interpret Results:** Decode the model's predictions to get the top 3 most likely labels for each image and display them.
- **Suggested Tools:** TensorFlow/Keras libraries.
- **Expected Outcome:** A Python script or notebook that can take any image as input and output the top predicted classifications.

Part 2: Intermediate Level (The Object Detector)

Focus: Finding the location of specific objects within an image.

Task 3: Where is the Object? (Object Detection)

- **The "Why":** It's not enough to know a person is in the frame; the security camera needs to know *where* they are. This task involves drawing a box around detected objects.
- **Description:** Use the state-of-the-art YOLO (You Only Look Once) model to detect multiple objects in a single image.
- **Step-by-Step Guide:**
    1. **Set up YOLO:** Download the pre-trained YOLOv3 or YOLOv4 model weights and configuration files.
    2. **Load Model:** Use OpenCV's DNN (Deep Neural Network) module to load the YOLO model.
    3. **Process Image:** Feed an image containing multiple objects (e.g., a street scene with cars, people, and traffic lights) into the model.
    4. **Draw Bounding Boxes:** Process the model's output to get the coordinates, class labels, and confidence scores for each detected object. Draw labeled boxes around them on the original image.
- **Suggested Tools:** Pre-trained YOLOv4 model files.
- **Expected Outcome:** A Python script that takes an image path as input and displays the image with bounding boxes drawn around all detected objects.

Task 4: Who is That Person? (Face Recognition)

- **The "Why":** The camera can now find people. The next step is to recognize *who* they are. Is it a family member or an unknown visitor?
- **Description:** Build a simple face recognition system to identify specific individuals in a set of photos.
- **Step-by-Step Guide:**
    1. **Create a Known Faces Database:** Use a library like **face_recognition**. Provide it with 1-2 images of a few "known" people and have it learn their face encodings.

2. **Load an "Unknown" Image:** Load a new image that contains one or more of the known people.
3. **Find and Identify Faces:** Locate all faces in the unknown image and compare their encodings to the known encodings.
4. **Label Faces:** Draw bounding boxes around the faces in the unknown image and label them with the correct names.

- **Suggested Tools:** Python's face_recognition library.
- **Expected Outcome:** A Python script that can identify and label known individuals in a group photo.

Part 3: Advanced Level (The Real-Time Analyst)

Focus: Applying vision skills to video and recognizing actions.

Task 5: The Live Feed (Real-Time Object Tracking)

- **The "Why":** Security cameras work with video, not just static images. This task involves applying your object detection skills to a live video stream to track objects from frame to frame.
- **Description:** Adapt your object detection model from Task 3 to perform real-time tracking on a pre-recorded video or live webcam feed.
- **Step-by-Step Guide:**
    1. **Video Input:** Use OpenCV to read frames from a video file or capture them from a webcam.
    2. **Process Each Frame:** In a loop, apply your YOLO object detection model (from Task 3) to every single frame.
    3. **Display Output:** Show the video stream in a window with the bounding boxes being drawn in real-time.
    4. **Performance Metric:** Calculate and display the Frames Per Second (FPS) to measure the performance of your system.
- **Suggested Tools:** OpenCV, Pre-trained YOLO model.

- **Expected Outcome:** A Python script that opens a video stream and displays real-time object detection, with bounding boxes tracking objects as they move.

Task 6: What Are They Doing? (Action Recognition)

- **The "Why":** The ultimate "smart" camera doesn't just see a person; it understands their actions. Is someone waving for help, or just walking by? This is the most advanced challenge.
- **Description:** Build a model that can classify simple human actions from short video clips.
- **Step-by-Step Guide:**
    1. **Data Preparation:** Use a dataset of video clips labeled with actions (e.g., "waving," "clapping," "walking").
    2. **Feature Extraction:** For each video, you might extract features from each frame using a pre-trained CNN (like in Task 2).
    3. **Sequence Modeling:** Feed these sequences of features into a recurrent neural network (**LSTM**) to learn the temporal patterns of each action.
    4. **Train and Classify:** Train the model and then use it to classify new, unseen video clips.
- **Suggested Dataset:** UCF11 Action Dataset (a smaller, manageable dataset).
- **Expected Outcome:** A Jupyter Notebook detailing the process of building and training an action recognition model. The final output should be a function that takes a video clip and predicts the action being performed.