Assignment - Computer Vision Engineer

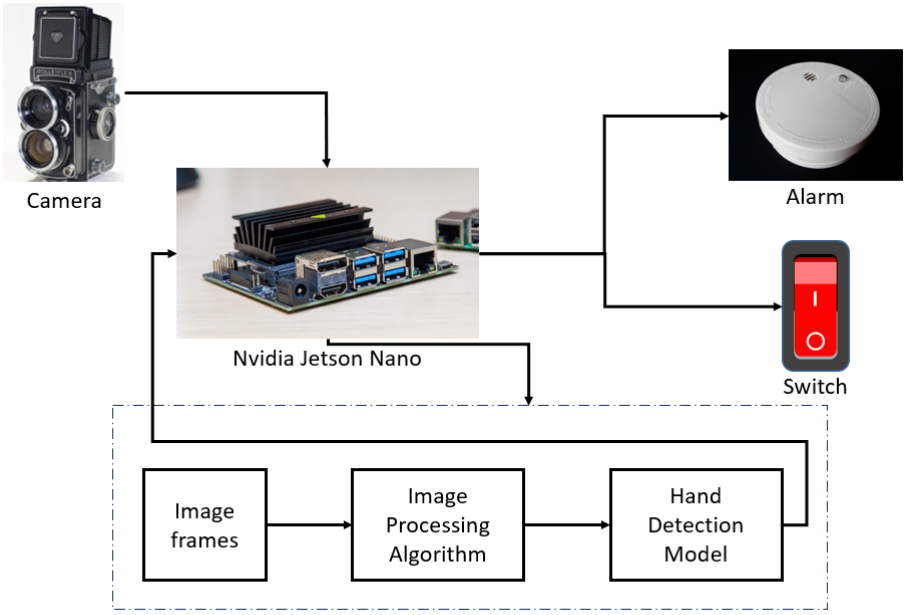
ELANSOL TECHNOLOGIES

Project 1: Hand Detection for Safety in Machine Operation

Objective: Develop a vision system to detect and prevent accidents by identifying human hands in proximity to critical machine parts.

1. System Architecture:

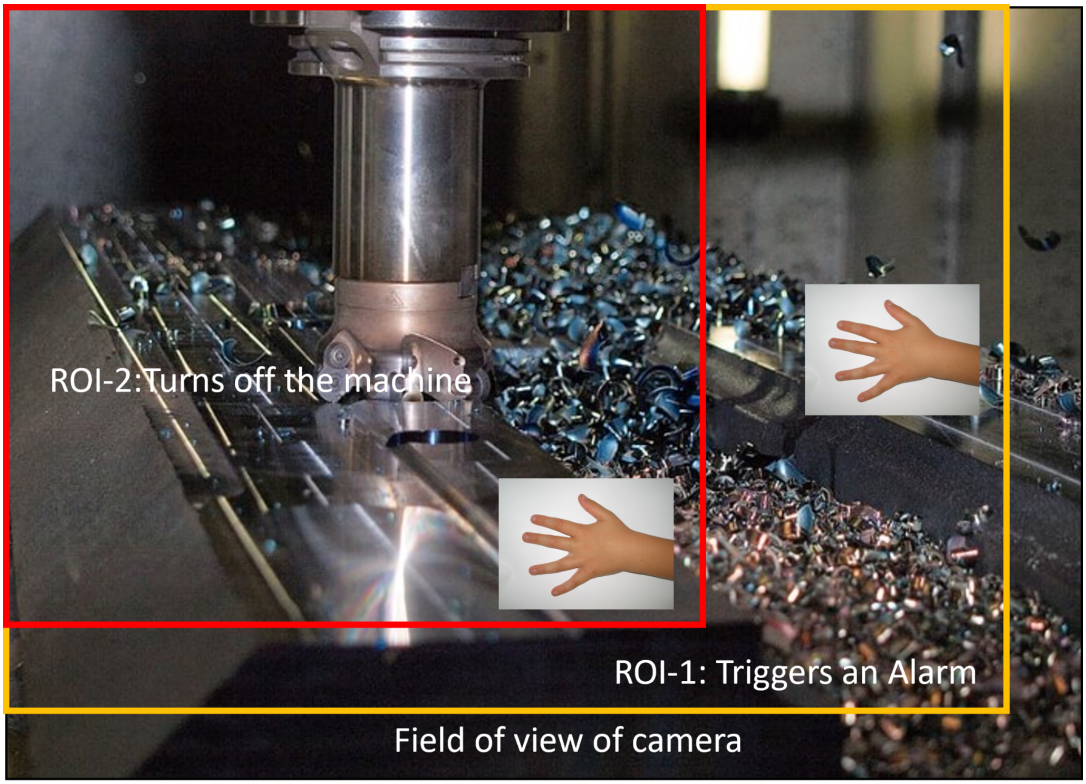
1. **System Architecture:** The vision system will consist of hardware and software components. The hardware includes a camera to record a live video stream and an Nvidia Jetson Nano for processing the video and running the hand detection model. We will also require an alarm system and an automatic switch that can be triggered when any hand is in near sight of the machine. The software components include required libraries, image preprocessing algorithms, a hand detection algorithm, and a deep learning model for hand detection.
2. **Utilization of Nvidia Jetson Nano:** The Nvidia Jetson Nano will be utilized as the processing unit for the vision system. Its powerful GPU capabilities and low power consumption make it ideal for real-time video processing. The Jetson Nano will run the hand detection algorithm and deep learning model to identify human hands in the video stream.
    a. In this system, Nvidia Jetson Nano is the central processing unit for real-time image processing and hand detection.
    b. Nvidia Jetson Nano is capable of running multiple neural networks in parallel. It can be used for image classification, image segmentation, object detection, etc. The GPU in Nvidia Jetson Nano will provide real-time functionalities for these applications.
    c. It will run the hand detection algorithm on the captured images or videos to identify human hands in the proximity of critical machine parts using GPU acceleration.
    d. Alarm Trigger/ Power off: When the Jetson Nano detects a human hand in proximity to critical machine parts, it will activate the alarm system or turn off the machine depending on the proximity of the hand.



An Overview of Hand Detection for Safety in Machine Operation

2. Image Processing and Hand Detection:

1. **Image Processing:** The captured image frames may contain noise, variations in lighting conditions and other artifacts that can affect hand detection. To enhance the quality of the image frame they will undergo preprocessing steps such as resizing, noise reduction, and contrast enhancement to improve the quality and reduce noise from the image. The feature extraction step will be handled by the deep learning model used for hand detection.
    a. Image resizing: Resizing the images to a standard resolution will ensure faster preprocessing and reduce computational overhead. We can use the resize function provided in OpenCV by experimenting with interpolation techniques.
    b. Noise Reduction: Techniques like Gaussian Blurring or median filter can be used to reduce the noise from captured image frames. OpenCV provides these functionalities.
    c. Contrast Adjustment: Histogram Equalization techniques such as Contrast Limited Adaptive Histogram Equalization (CLAHE) or other Adaptive histogram equalization techniques can be used towards this end. Again OpenCV provides these functionalities.
2. **ROI selection:** Based on the position of our mounted camera, we will have certain regions that are more crucial. We can predefine it in our image and if our hand detection model detects a hand in this region we can start the alarm or if it is in the more crucial region we can even turn off the machine.



Example Illustration of how an ROI can be utilized. We can predefine two ROIs based on the camera field of view and if we detect hands inside these ROIs we trigger the alarm or turn off the machine depending on the ROI.

3. **Hand Detection Model:** So we have two options for this section:
    a. Object Detection model: A model such as YOLO (You only look once) that detects the position of the object (Hand in our case) gives the position of the hand. Based on this prediction, we can find out if the hand is in critical ROIs or not.
    b. Hand Classification model: From the mounting position of our camera, we know the ROIs, we can crop out the ROIs from the image and send it to an image classification model (that might be lightweight than the YOLO model) and know if Hands are present inside the critical ROI.

Both models can work in real time with the help of Nvidia Jetson Nano. The advantage of YOLO algorithm is that we can track hand movement in real-time. However, the classification model will only be able to predict whether a hand is present inside our crucial ROIs with lesser computation and more speed. There are several other methods for Object detection such as RCNN, Fast RCNN, Faster RCNN, etc. but they are not fast like YOLO so for hand detection algorithm we should choose YOLO over these methods. For the classification model, we have a lot of CNN-based (Transformer based models will be too heavy to run on a small processor like Nvidia Jetson Nano) models such as Resnet, Efficient Net etc. We should experiment with different models for classification accuracy, efficiency and time taken.

3. AI Training and Development:

First off we can start with a pretrained model on some big dataset (e.g. MS COCO dataset for YOLO or ImageNet for classification model). Then we plan to again pre-train it on a publicly available hand detection dataset (e.g. HaGRID Dataset). After that, we fine-tune it on the data collected and annotated by us.

1. **Data Collection Process:** This process is crucial for any machine learning-based project. We should ensure that the images collected in this process should be diverse and representative of the dataset of images containing hands in proximity to critical machine parts, covering a variety of lighting conditions, hand orientation, and potential occlusions. Also, for robustness, the images should include objects with different backgrounds, object sizes and hand positions. If possible we should also collect the images from the mounted camera with the crucial ROI that will help our model to give more accurate results. Also, the dataset should contain images with hands and without hands.
2. **Data Annotation Process:** So, here we are planning to experiment with two ML models: 1. Hand detection algorithm (YOLO) and 2. Image Classification model (Resnet18). The annotation for the hand detection algorithm will define the bounding boxes around the hands in images. Similarly, annotations for the classification model will only require if the hand is present in the image. We can use LabelImg online tool for creating bounding box annotations of hand-detection datasets.
3. **Training Pipeline:**
    a. Data Preprocessing: In this step, we resize the dataset to a particular image size then normalize the dataset and perform data augmentations before feeding the dataset to the AI model. We can use linear scaling or z-score normalization, for augmentation we can use rotation, scaling, flipping, addition of noise (gaussian and uniform) and brightness adjustment. Along with this we also split our dataset into train validation and test set to evaluate our model on unseen dataset.
    b. Selection of Deep Learning Framework and Model: We will first choose a suitable Deep Learning framework (Pytorch or TensorFlow) for the training process.
    For model selection, we should experiment with different versions of the YOLO object detection model and different classification models. We will choose the models based on their performance on unseen dataset along with their real-world performance during system test time.
    c. We will use pre-trained models wherever applicable (YOLO pretrained on the COCO dataset and Resnet18 pretrained on ImageNet Dataset). We then again pre-train the model on some hand recognition datasets (HaGRID dataset). After that, we fine-tune the model on our annotated dataset and evaluate its performance.

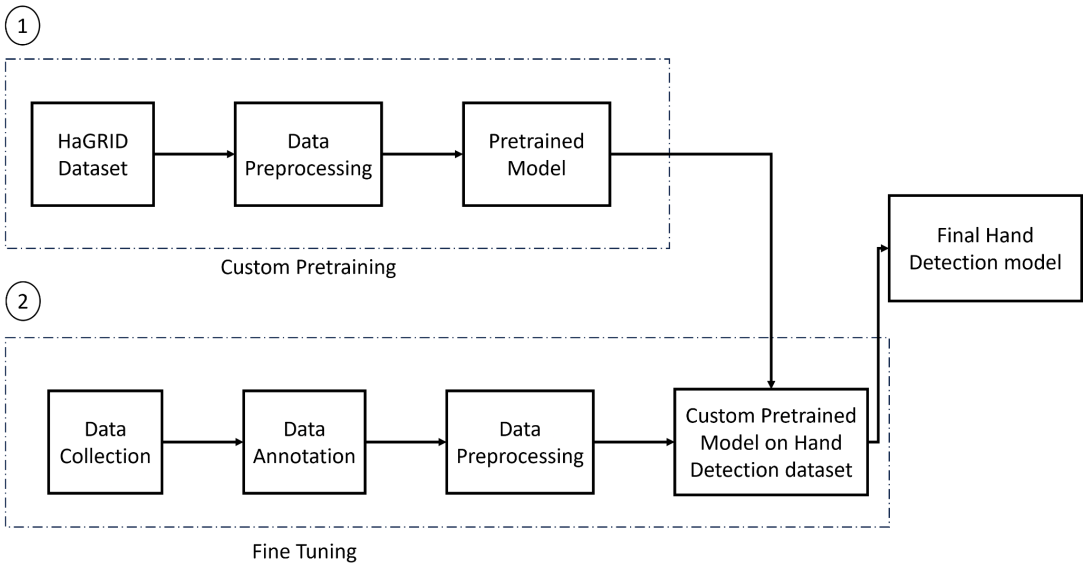


Illustration of the Training Pipeline

4. **Validation and Testing:** To validate the models we first evaluate their performance on unseen test data created in the Pre-processing step. After this to evaluate the model in the real world we will evaluate the best model's performance with a live camera input.

4. Deployment Planning:

1. Deployment Strategy: In this step, our main motive will be to integrate all the steps from taking input from the camera to buzzing the alarm or stopping the machine into one whole system. To achieve this we will integrate the camera for input, our hand detection algorithm and alam/ turning off sequence with Nvidia Jetson Nano.
2. Optimizations for real-time performance: To achieve real-time performance, we aim to perform optimization on the hand detection model with techniques like quantization and model pruning. We also aim to utilize parallel processing in Nvidia Jetson Nano to increase the processing time. For this we can utilize TensorRT which is an optimization and inference library developed by NVIDIA specifically for deep learning applications. It is designed to accelerate the inference performance of deep neural networks on NVIDIA GPUs. By using TensorRT, developers can achieve highly optimized and efficient deep learning inference on NVIDIA GPUs, making it an essential tool for deploying AI models in production environments.
3. Challenges and Limitations: Some of the challenges that might arise in the deployment process are:
    a. Real-time processing: Achieving real-time performance with a complex deep learning model on Nvidia Jetson Nano device may require some trade-off on model performance
    b. Compatibility issue between different hardware: There might be a possibility that the hardware interfaces and protocols are not well-matched, Necessary planning for this while selecting the hardwares is essential

c. False Alarms: It is crucial to minimize the false alarms to avoid desensitizing the users to genuine safety alerts. This is something that may require some more tuning of the model after deployment
d. Cost and Budget Constraints: Cost and budget should always be kept in mind while developing this project.

Project 2: Process Monitoring Using Video Analytics

Objective: Implement a vision system to monitor operator assembly processes and ensure the correct execution of steps.

1. System Architecture:

System Architecture: The vision system will consist of hardware and software components. The hardware includes a camera to record a live video stream and an Nvidia Jetson Nano for processing the video and running the object detection and process identification models. We will also require an alarm system or will trigger an alert or notification to notify relevant personnel on detecting a deviation or incorrect step. The software components include required libraries, image preprocessing algorithms, object detection algorithms to identify objects involved in the assembly process, a sequence-to-sequence or RNN-based model for process identification i.e. to recognize and classify each step in the assembly process based on the detected objects and analyze the temporal sequence of detected steps to ensure the correct order and execution.

2. Video Analysis and Process Identification:

1. **Approach to analyze assembly process from video footage:**
    a. Train Object Detection Model: Initially, an object detection model will be trained to identify the position of different components used for assembly in our specific case.

b. Object Detection during Assembly: During the assembly process, the trained object detection model will analyze each frame and identify the components present, along with their positions in the image.
c. Frame Vectorization: The frames containing detected components will be converted into vector inputs suitable for training sequential models like RNN or LSTM.
d. RNN/LSTM Training: Sequential models, such as RNN or LSTM, will be utilized with the vectorized frames as inputs. The output labels for these models will represent the current step or process being executed during assembly.
e. Pattern Verification: By using the RNN or LSTM outputs, it becomes possible to detect whether the assembly process follows the correct pattern or if any steps are incorrect.

2. **Use of video analytics and AI techniques to identify and track the individual steps in the process:** By combining object detection and sequence-to-sequence models, video analytics and AI techniques can proficiently identify and track individual steps in the assembly process. For object detection, the YOLO algorithm is employed to identify objects in each video frame, and the model is trained on assembly-specific images or frames. Subsequently, the detected objects and bounding boxes are transformed into feature vectors, including size, shape, and object class. Utilizing sequence-to-sequence or RNN-based models, assembly steps are identified and classified using these feature vectors for frame-wise predictions during training and validation on labeled datasets.

3. **Algorithm:**
    a. Object Detection model: A model such as YOLO (You only look once) that detects the position of the object (Different components used for assembly in our case) gives information about which objects are present in a particular assembly process and their position in the form of a bounding box. We can feed this information to the process identification model to identify the process sequence.
    b. Process Identification model: RNNs or LSTM networks can be used as process identification models. The input sequence (detected components) is converted into a fixed-size representation called the context vector. RNN takes the context vector and generates the output sequence of predicted steps.

    Both models can work in real time with the help of Nvidia Jetson Nano. The advantage of YOLO algorithm is that we can track hand movement in real-time. For the process identification model, we have simple sequence to sequence models such as RNN, LSTM, GRU, etc which can easily run on a small processor like Nvidia Jetson Nano. We should experiment with different models for classification accuracy, efficiency and time taken.

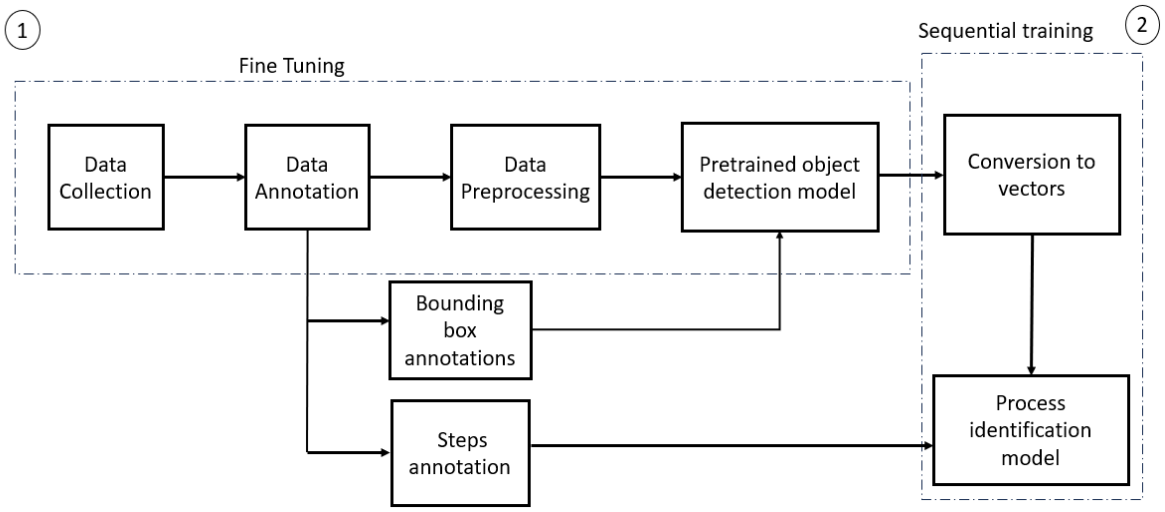


Illustration of training pipeline

3. AI Training and Development:

First off we can start with a pretrained model on some big dataset (e.g. MS COCO dataset for YOLO). After that, we fine-tune it on the data collected and annotated by us.

1. **Data Collection Process:** The developed model should be able to detect all the objects needed for assembly. Since the assembly components are not that common to find in everyday life, the dataset has to be composed of videos captured by us. In order to have a big enough dataset, we can capture videos of the objects while moving the camera around the object and separate their frames by placing and recording objects in different places and poses. This ensures that the dataset contains photos of the part of interest, from many different angles and backgrounds. Videos should be captured under various lighting conditions. Also, if possible, the dataset should contain videos with operators assembling the components from the camera we are planning to use.
2. **Data Annotation Process:** So, here we are planning to experiment with two ML models: 1. Hand detection algorithm (YOLO) and 2. Sequential model (RNN or LSTM). The annotation for the component detection algorithm will define the bounding boxes around different components in images. Similarly, annotations for the sequential model will only require the labeled steps at each time step. We can use LabelImg online tool for creating bounding box annotations of object detection datasets and similar tool for labeling steps at each time for Process identification.
3. **Training Pipeline:**
    a. Data Preprocessing: Same as hand detection project
    b. Selection of Deep Learning Framework and Model: We will first choose a suitable Deep Learning framework (Pytorch or TensorFlow) for the training process. For model selection, we should experiment with different versions of the YOLO object detection model and different sequential models. We will choose the

models based on their performance on unseen dataset along with their real-world performance during system test time.

c. We will use pre-trained models wherever applicable (YOLO pretrained on the COCO dataset). We can also use LSTM in place of RNN to solve the vanishing gradient issue and tackle long-range dependencies. Encoder-decoder networks can also be used, but our problem is not very complex. Hence, RNN or LSTM can be a good choice, as they are less computationally expensive. After taking a pre-trained model for object detection, we fine-tune the model on our annotated dataset and evaluate its performance.

4. **Validation and Testing:** We will first evaluate our models with held-out test data that we have created in the Data Preprocessing step with different metrics such as process identification accuracy for RNNs or LSTMs and mean average precision(mAP) for the Object detection model. After we have selected our best model we will test them in an online setting to evaluate them in the real world.

4. Deployment Planning:

- Similar to hand detection project