**Training a Keypoint Estimation Machine Learning Model**

**Github Repository:** [**https://github.com/manavchouhan115/KeypointEstimation**](https://github.com/manavchouhan115/KeypointEstimation)

**Weekly Updates**

**Week 1:**

1. Downloading and Understanding the data from the dropbox.
2. Wrting a python script to extract the worker frames count and timeSpent on all the frames throughout the data.
3. Code Link : <https://github.com/manavchouhan115/KeypointEstimation/blob/main/DataPreprocessing/extractWorkerIdFramesandTimeSpent.py>
4. Implementing a basic OOTB Yolo model with few frames from the dataset using ultralytics/yolov5. It is a multi-class classification model.
5. Ultralytics YOLOv5 is an open-source object detection model developed by Ultralytics, based on the YOLO (You Only Look Once) family of models. YOLOv5 is highly regarded for its real-time object detection capabilities, speed, and performance. It is widely used for detecting objects in images and videos and can classify multiple objects in a single pass with high accuracy.
6. When a frame from the dataset is processed by the YOLOv5 model (as given below) , it frequently identifies the sticks as baseball bats, likely due to similarities in shape and size between the two objects.
7. The model’s performance further degrades when the input image is blurred or contains multiple sticks, leading to significant challenges in detecting and classifying the objects correctly. In such cases, the model struggles to distinguish between individual sticks or may fail to detect them altogether, highlighting limitations in handling visual noise or complex scenes with overlapping objects.
8. These issues suggest that the model might benefit from further fine-tuning or improvements in its ability to handle variations in image quality and object density.



**Week 2:**

1. Exploratory Data Analysis of the videos stills and annotations.
2. Finding Data Inconsistencies. (Total 1132 images, 1101 labels)

Potential Inconsistencies

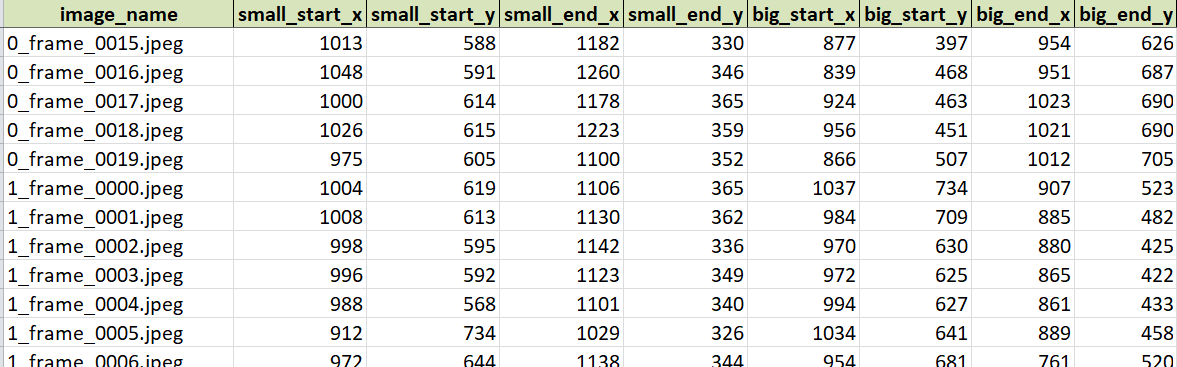
* There are some incomplete labels - example - For output 9 - there are total 20 frames captured but only 12 of them are labelled. 8 are missing from the json file. **(Unmarked frames to be discarded)**
* Some frames don't have any of the sticks visible - example- 10/frame18. So, there are no vertices for them.
* As we discussed before that if there is only a single stick it should be labeled as small. but here are some cases in which even though there is only single stick, it is marked as big.

1. Copy all the images to target data folder and rename the images to unique distinguishable name.

Code link: <https://github.com/manavchouhan115/KeypointEstimation/blob/main/DataPreprocessing/LoadData.py>

1. Extract the data of stick label (small, big), vertices and store them in a model readable form (DataFrame).
2. Extracted the data into a DataFrame containing 9 columns. One for Image name and other 8 co-ordinates for end points of each big and small sticks.

Code Link: <https://github.com/manavchouhan115/KeypointEstimation/blob/main/DataPreprocessing/transform_labels.py>



1. Train-Test Split: For an image classification problem with around 1000 images using a the train-test split helps assess how well the fine-tuned model generalizes. Typically, an 80/20 split is used, where 80% of the images are for training (fine-tuning the pre-trained model), and 20% are for testing. But as we are using a pretrained model and will do fine-tuning on that model for our classification task 50-50 split can also be used to start with. It will give us more data for the validation set.
2. Image Classification models :
3. **EfficientNet**: High accuracy but requires larger computational power.
4. **MobileNet**: Trades off between accuracy and efficiency
5. Yolov8 : Object detection model. Can be fine-tuned for stick labels.

**Week 3:**

1. Preparing the Data for the models.

* The Image Classification models such as MobileNet, ResNet, Efficient Net uses a particular image resolution as an input.
* For MobileNet and ResNet default input size is 224x224
* EfficientNet models scale the input size based on the model version (EfficientNet-B0 to B7). The typical Input sizes ranges from 224x224 to 600x600
* **Training**: When training models, sticking to the default input sizes is generally recommended as they have been optimized for those sizes. And since we are using Transfer learning it increases efficiency.
* Each model expects 3-channel (RGB) images. we should convert them to 3 channels before feeding them into the model.

Code link:

1. Trying out the prepared dataset on various models. Tried out adding custom layers and using various batch sizes. Best Results are shown below
2. Using ImageNet model

*# Transfer Learning Model - Use MobileNetV2*

*base\_model = MobileNetV2(input\_shape=(224, 224, 3), include\_top=False, weights='imagenet')*

*base\_model.trainable = False # Freeze the base model*

*# Add custom layers*

*x = base\_model.output*

*x = GlobalAveragePooling2D()(x)*

*x = Dense(1024, activation='relu')(x)*

*x = Dropout(0.5)(x)*

*# Output layers for classification and stick coordinates regression*

*stick\_classification = Dense(4, activation='softmax', name='stick\_classification')(x)*

*stick\_coordinates = Dense(8, activation='linear', name='stick\_coordinates')(x)*

loss={'stick\_classification': **'categorical\_crossentropy'**, 'stick\_coordinates': **'mean\_squared\_error'**},

metrics={'stick\_classification': **'accuracy'**, 'stick\_coordinates': **'mae'**}

**epochs=10, batch\_size=32**

Epoch 10/10

**28/28** ━━━━━━━━━━━━━━━━━━━━ **71s** 2s/step - loss: 2069.6689 - stick\_classification\_accuracy: 0.7005 - stick\_coordinates\_mae: 35.9749 - val\_loss: 2204.3464 - val\_stick\_classification\_accuracy: 0.7321 - val\_stick\_coordinates\_mae: 37.1083

Though it gets a good classification accuracy, the mae is around 40 which gives a inaccurate result.

1. Using ResNet50 model – A larger 50 layes model.

*base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))*

*base\_model.trainable = False # Freeze the base model*

*# Add custom layers*

*x = base\_model.output*

*x = GlobalAveragePooling2D()(x)*

*# Output layers for classification and stick coordinates regression*

*stick\_classification = Dense(2, activation='sigmoid', name='stick\_classification')(x)*

*stick\_coordinates = Dense(8, name='stick\_coordinates')(x)*

Epoch 10/10

**55/55** ━━━━━━━━━━━━━━━━━━━━ **301s** 5s/step - loss: 3804.7727 - stick\_classification\_accuracy: 0.6669 - stick\_coordinates\_mae: 50.3552 - val\_loss: 3600.0256 - val\_stick\_classification\_accuracy: 0.543 - val\_stick\_coordinates\_mae: 49.3350

The accuracy in a larger model reduces to 66 % and and mae gets to 50.

Needs fine tuning with different parameters for to increase accuracy

1. Both the models took around **40 mins** to train on Colab T4 GPU.
2. Need to work further on fine Tuning the model to get better results.

**Week 4:**

1. **Yolov8 model** - YOLOv8 is the latest iteration in the You Only Look Once (YOLO) family of object detection models, designed for superior speed and accuracy in computer vision.
2. It offers better performance in real time applications.
3. YOLOv8 can detect multiple classes of objects within an image or video frame.
4. Model Variants like YOLOv5, YOLOv8 offers different model sizes, such as YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), and YOLOv8l (large). These variants balance the trade-off between computational cost and accuracy.
5. I have trained our dataset on **Yolov8s (small) model** and it has provided the best accuracy so far.
6. Yolo is generally a bounding box algorithm. I have tried to fine tune it with outputting heatmap lines for identifying sticks.
7. Yolo models are available through **Ultralytics** library provides simple APIs to train.
8. Parameters used for training:

epochs=50, # Number of epochs

imgsz=640, # Image size

batch=16, # Batch size

Optimizer – Adam

Learning rate = 0.01

1. The training part took around 1 hr to run on Google colab for 50 epochs.
2. Results for the training are as follows

Ultralytics YOLOv8.2.98 🚀 Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4, 15102MiB)

Model summary (fused): 168 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs

Class Images Instances Box( P R mAP50 mAP50-95):

all 234 437 0.683 0.526 0.6 0.369

small\_stick 215 215 0.63 0.435 0.507 0.301

big\_stick 222 222 0.737 0.617 0.694 0.436

1. The results are not promising enough for real time applications. Need to fine tune model further.
2. Result example



1. As you can see above The model is able to classify the sticks as big and small and also able to identify its coordinates correctly.
2. Limitations:
3. The accuracy as the model is low. Not able to identify all the images correctly.
4. Misclassifies small and big sticks opposite sometimes
5. If the whole stick is not clearly visible. It fails to identify it at all.
6. For lower quality images or if the background is bright the accuracy goes down further.
7. Is identifying more than 3 sticks too if its visible.
8. Not really scaling efficiently on real time.