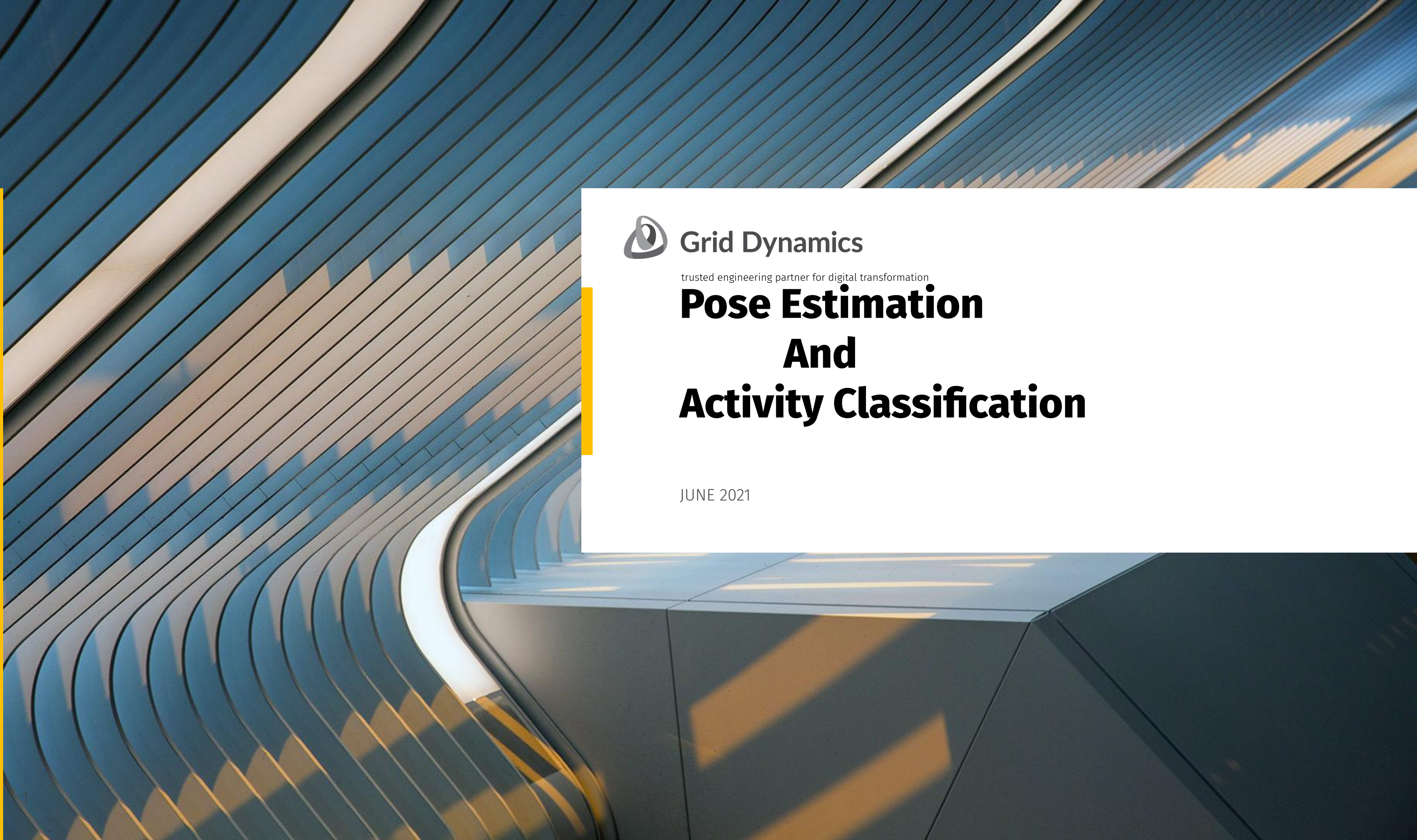


www.griddynamics.com

Grid Dynamics

trusted engineering partner for digital transformation

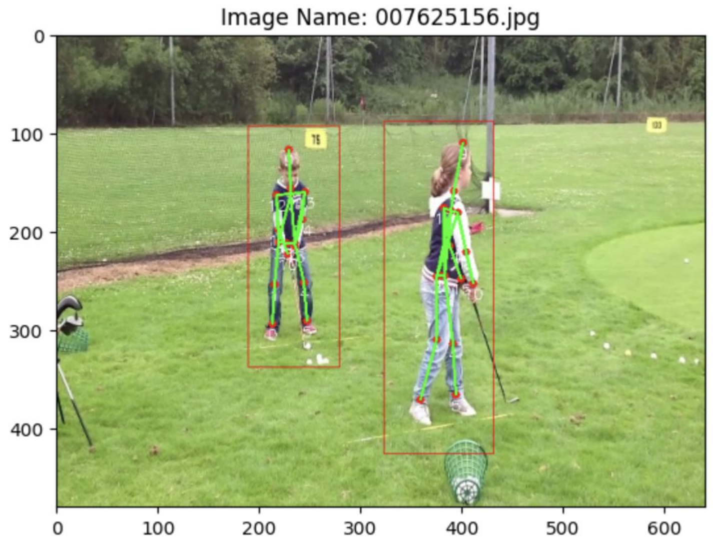
Pose Estimation And Activity Classification

JUNE 2021

Exploring the dataset

Our dataset contains images of humans doing various activities, the type of activity being performed and the respective keypoints of the various joints of the humans present in the images and the visibility of the respective keypoints.

1. Total Images: 24987(valid + invalid)
2. Number of valid images: 17408(some images don't have annotations).
3. Number of Joints: 16
4. Visibility Type:
   - 4.1. 0: Joint not visible.
   - 4.2. 1: Joint is visible.
5. Keypoints Co-ordinates:
   - 5.1. x: Location of the x-coordinates for different joints.
   - 5.2. y: Location of the y-coordinates for different joints.



Example->Activity(Sports)

Exploring the bad annotations and how they are handled

The various scenarios for bad annotations are:

1. **Negative Co-ordinates:** Some annotations have negative co-ordinates for the keypoints.
   1.1. **How its dealt with:** If either x or y co-ordinates of a keypoint have negative co-ordinates both the co-ordinates are replaced as (0, 0) and their visibility is marked as 0 as well.
2. **Out of bounds Co-ordinates:** Some annotations have co-ordinates which are beyond the resolution of the images, like if the resolution is (720, 1280) and (y co-ordinate > 720) or (x co-ordinate > 1280).
   2.1. **How its dealt with:** If either of the co-ordinates are beyond the resolution of the image, both are replaces as (x, y) = (0, 0) and their visibility is marked as 0 as well.
3. In this dataset we have to calculate bounding box by ourself, which we do by calculating the min,max of (x,y). There are two cases where we need to add some extra padding.
   3.1. **Missing Head-Top Keypoint:** If an image has a missing head-point(x, y) = (0, 0)
      3.1.1. **How its dealt with:** We introduce an extra padding(y_min=y_min - padding) within the bounding box to compensate for the loss in bounding box caused by missing head-top keypoint.

3.2. **Both leg-ankle Keypoints Missing:** If an image has both leg-ankle missing or (x, y) = (0, 0).

3.2.1. **How its dealt with:** We introduce an extra padding(y_max=y_max + padding) within the bounding box to compensate for the loss in bounding box caused by missing leg-ankle Keypoint.

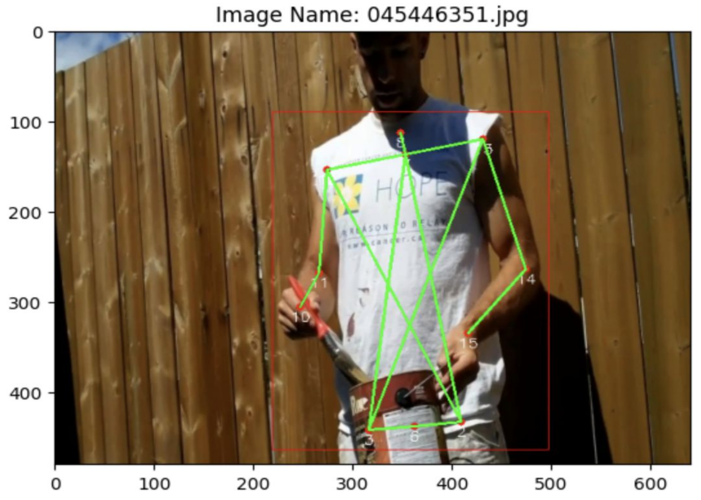
4. **Empty(cat='[]') pose category:** If an image does have the annotation for the kind of activity the human is performing in the given image.

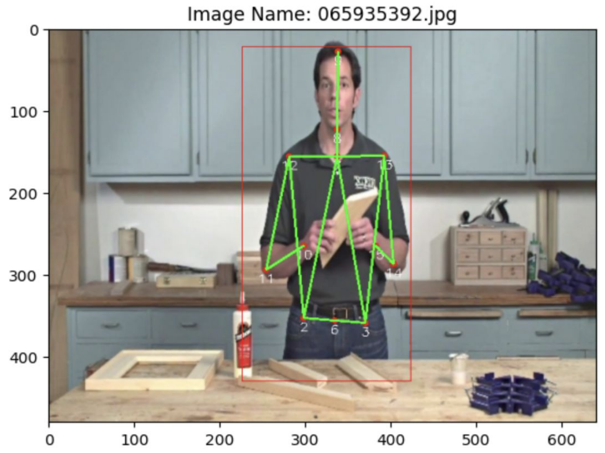
5.1. **How its dealt with:** We simply ignore such images, as they can't be fed into the model

6. **Images with no annotations:** If an image does have the annotation we simply ignore it.

Finally, after all such filtering we are left with the 17,408 images out of a set of 24,987 images.

Visualising some of the bad cases

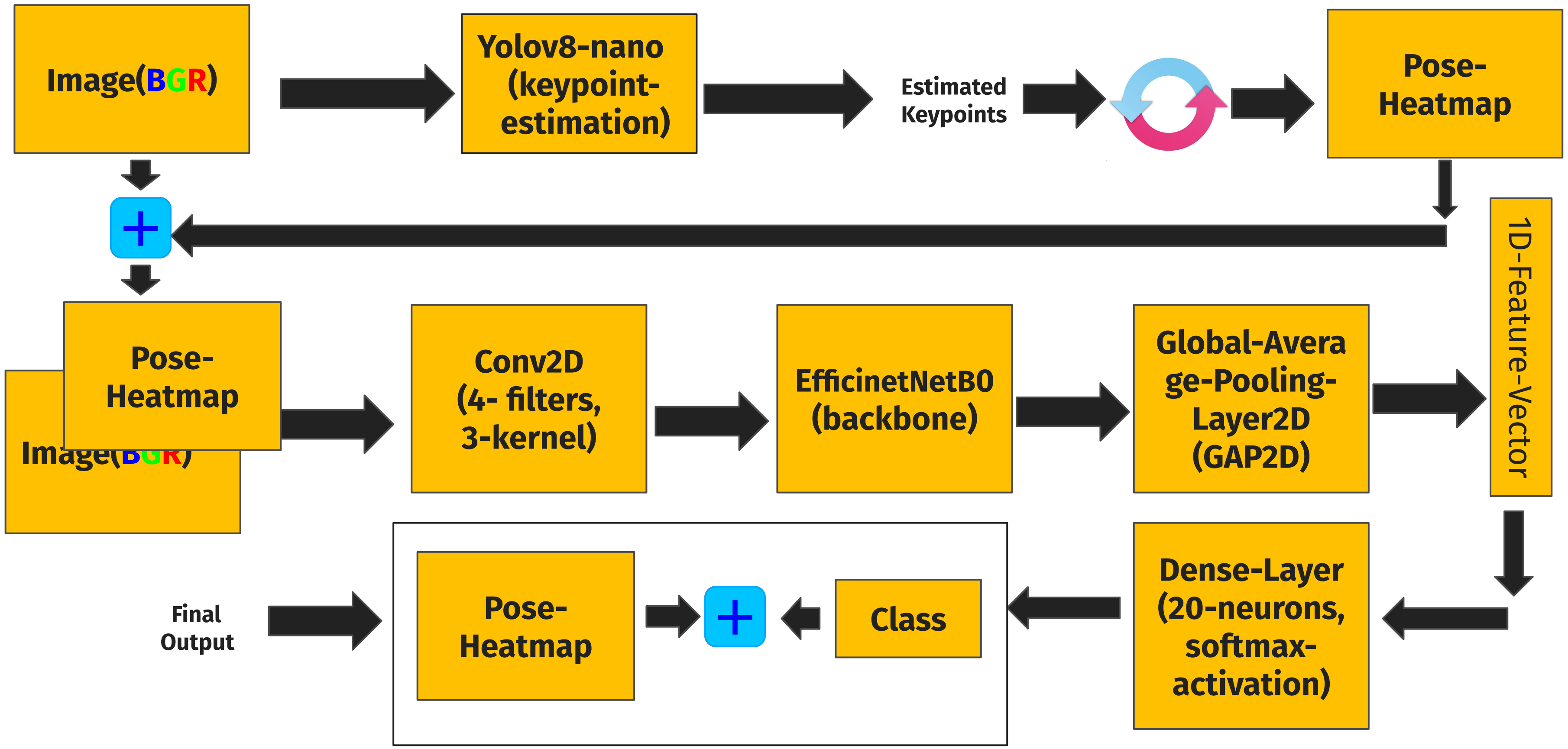




Missing Head-Top: Notice the extra-padding(bbox extends a little beyond the keypoint 8).

Both leg-ankle missing: Notice the extra padding(bbox extends a little below the keypoints 2,6,5).

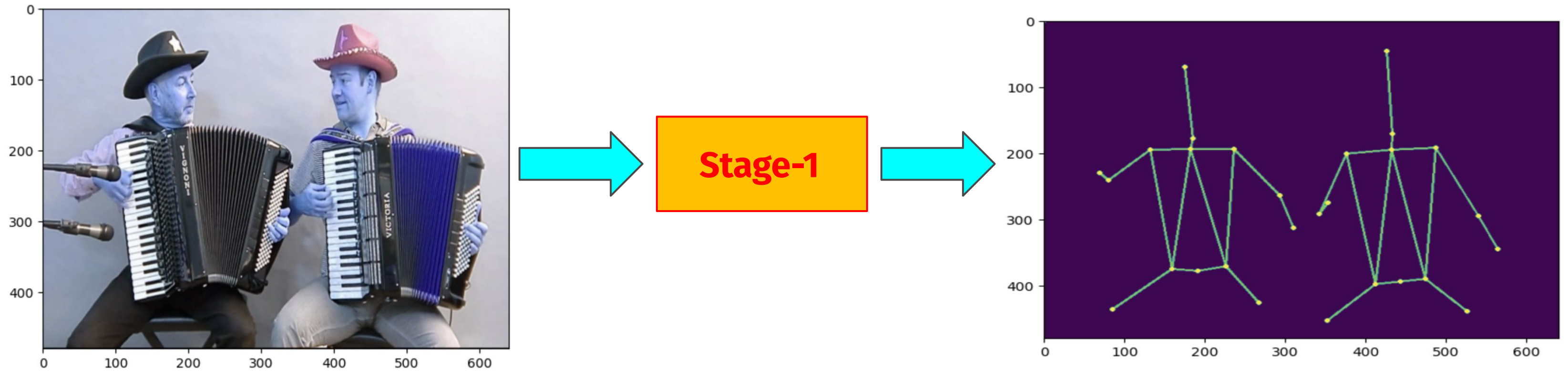
Model Architecture


Image(BGR)
Yolov8-nano (keypoint-estimation)
Estimated Keypoints
Pose-Heatmap
+
Pose-Heatmap
Image(BGR)
Conv2D (4- filters, 3-kernel)
EfficinetNetB0 (backbone)
Global-Average-Pooling-Layer2D (GAP2D)
1D-Feature-Vector
Dense-Layer (20-neurons, softmax-activation)
Class
+
Pose-Heatmap
Final Output

Architecture Explanation

Our model has two stages. Let’s discuss briefly:

1. **Pose-Estimation-Stage:** In the diagram on the previous page, this stage is represented in row-1, this stage basically uses yolov8-nano model to estimate the keypoints of humans in the image, using these keypoints we draw a pose-heatmap.



Image(480, 640, 3) **Pose-Heatmap(480,640,1)**

2. **Pose-Classification Stage:** In the model architecture, this stage is represented by row-2, 3. In this stage we take the input image(480,640,3) and pose-heatmap(480,640,1) and stack them on top of of each-other to create a (480,640,4) tensor, which is then fed to two conv2D layers with 4, 3 filters respectively and then its fed to EfficientNetB0 backbone model which is used as a feature extractor and generates (15, 20, 1280) tensors, that is, 1280 feature-maps of dimensions (15,20), which is then passed through a GloabalAveragePooling2D(GAP2D) to compress the feature information to a 1D vector(1280), this is then fed to a dense layer with 20 neurons with softmax-activation for pose-classification.

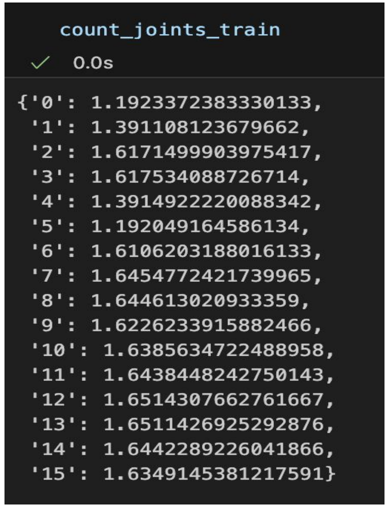
How the data is formatted and saved

- Firstly, all the images are resized to (480,640,3).
- After the stage-1 generates the pose-heatmap, we stack the information of image and pose together to produce(480,640,4) tensors.
- These tensors are are saved in '.npy' format for faster storage and retrieval, also the corresponding pose-labels are saved in '.txt' format.
- Each image has a corresponding '.npy' file as well as a '.txt' file for sample-info and corresponding label.
- Also note that when we generate pose-heatmaps, since multiple people can be present in a single image(say 4-6 people), we only take the pose of the center-most person and the person who is farthest apart from the center-most person, since if we take all the person, say there are 4 people, then this will lead to a lot of overlap and clutter in the pose-heatmap, which might confuse the model.

Amount of data Used and its distribution

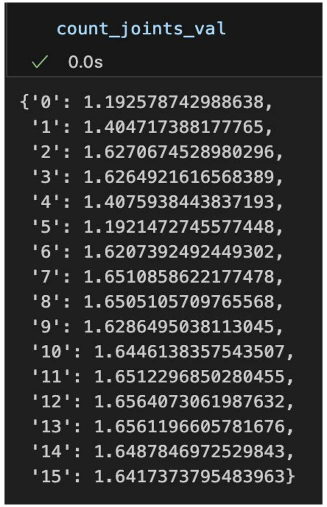
Let’s discuss the amount of data used for training and its distribution.

1. For training of yolov8(stage-1) we have used 2400 images in train_set and 1200 images in val_set.
2. The distribution of joints in both the sets in almost similar.



```
count_joints_train
✓ 0.0s
{'0': 1.1923372383330133,
 '1': 1.391108123679662,
 '2': 1.6171499903975417,
 '3': 1.617534088726714,
 '4': 1.3914922220088342,
 '5': 1.192049164586134,
 '6': 1.6106203188016133,
 '7': 1.6454772421739965,
 '8': 1.644613020933359,
 '9': 1.6226233915882466,
 '10': 1.6385634722488958,
 '11': 1.6438448242750143,
 '12': 1.6514307662761667,
 '13': 1.6511426925292876,
 '14': 1.6442289226041866,
 '15': 1.6349145381217591}
```

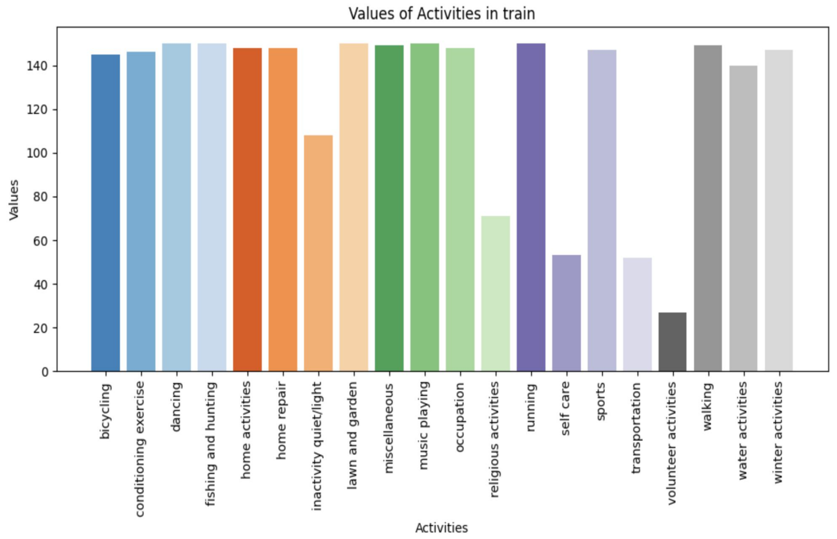
Average number of joint per image

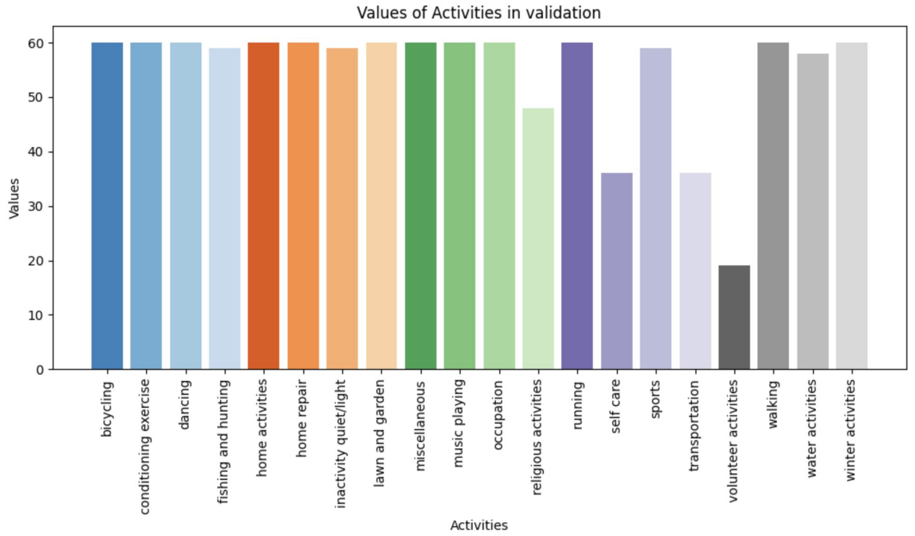


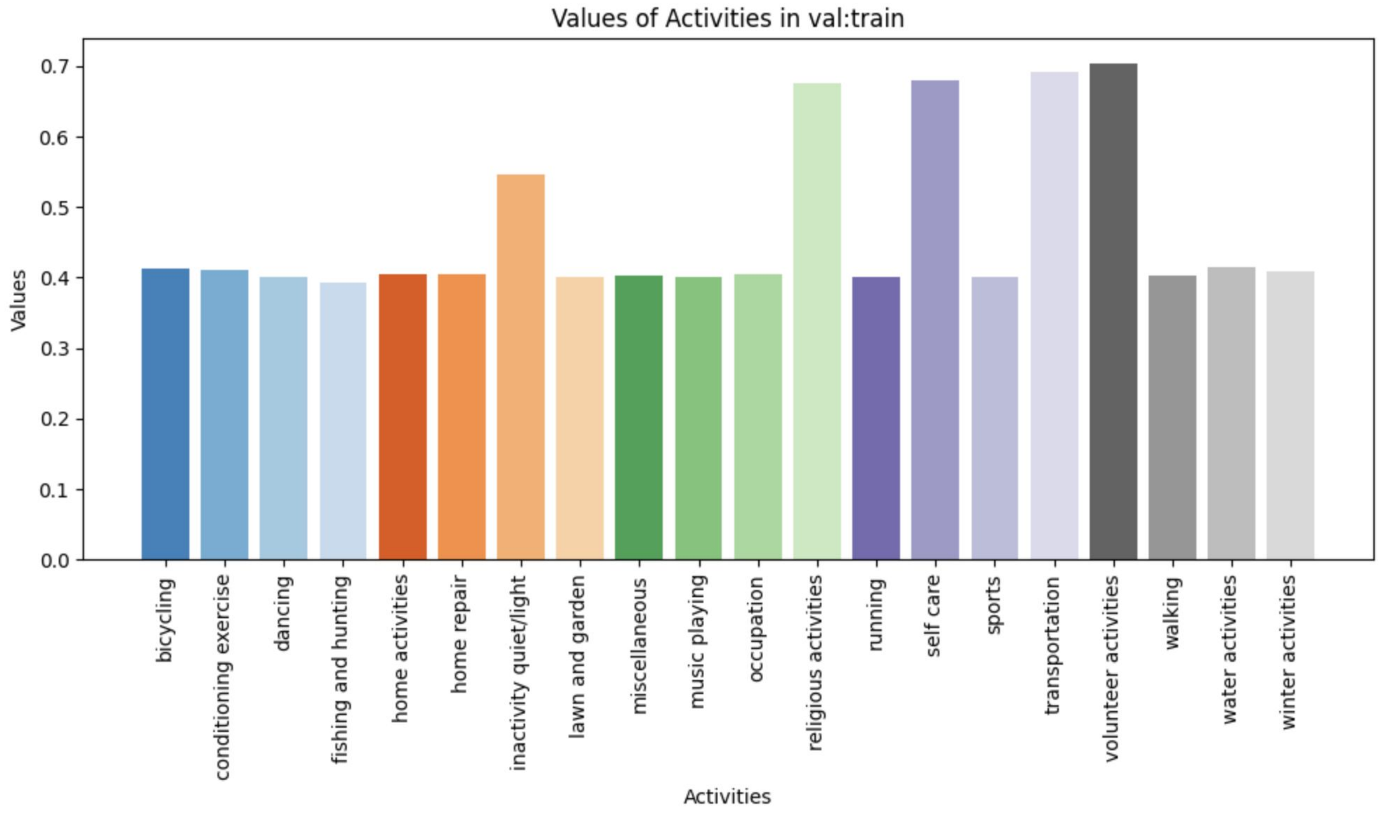
```
count_joints_val
✓ 0.0s
{'0': 1.192578742988638,
 '1': 1.404717388177765,
 '2': 1.6270674528980296,
 '3': 1.6264921616568389,
 '4': 1.4075938443837193,
 '5': 1.1921472745577448,
 '6': 1.6207392492449302,
 '7': 1.6510858622177478,
 '8': 1.6505105709765568,
 '9': 1.6286495038113045,
 '10': 1.6446138357543507,
 '11': 1.6512296850280455,
 '12': 1.6564073061987632,
 '13': 1.6561196605781676,
 '14': 1.6487846972529843,
 '15': 1.6417373795483963}
```

Average number of joints per image

3. For feeding the data to stage-2, we have used 2536 images in train_set and 1089 images in val_set.
4. We have splitted the data in ratio of train:total = 0.6 and val:total=0.4 and ensured that the distribution of images is same in both the sets.






Values of Activities in val:train
Values
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
bicycling
conditioning exercise
dancing
fishing and hunting
home activities
home repair
inactivity quiet/light
lawn and garden
miscellaneous
music playing
occupation
religious activities
running
self care
sports
transportation
volunteer activities
walking
water activities
winter activities
Activities

Training Process

The training of the model is a two stage process.

1. **Step-1:** First we train the yolov8-nano model(stage-1) to estimate the keypoint co-ordinates of the humans in the images.
2. **Step-2:** Secondly we then make inferences on the model obtained from stage-1 to generate the keypoint co-ordinates, which is used to generate the pose-heatmap, which is combined with image and then fed to stage-2 of the architecture. Thus the training of stage 2 is done after we have trained the model in stage-1, because stage-2 uses the inferences made by stage-1 as input to train itself.

Training Hyperparameters Used

1. **Stage-1:**
   - 1.1. Epochs: 45(25+20)
   - 1.2. Batch_Size: 64
   - 1.3. Freeze: 21(freeze the first 21 layers of yolov8-nano model)
   - 1.4. Cos_lr: True(using a cosine learning rate scheduler)
   - 1.5. Dropout: 0.1 (to prevent overfitting)
   - 1.6. Close_mosaic: 20(close the mosaic before the last 20 epochs)
   - 1.7. hsv_h: 0.02
   - 1.8. hsv_s: 0.7
   - 1.9. hsv_v: 0.5

2. **Stage-2:**
   - **2.1.** Epochs: 25
   - **2.2.** Optimizer: Adam

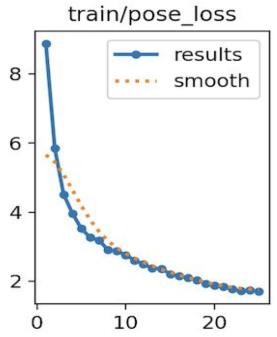
Metric Used

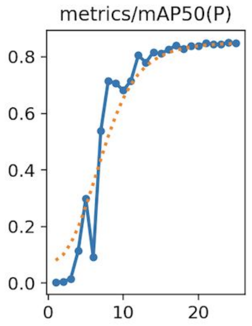
1. **Stage-1:**
   - **1.1. mAP50**(mean-average precision at 0.5 threshold)
   - **1.2. mAP50-95**(considering thresholds between 0.50-0.95 at 0.05 step and then averaging them)
   - **1.3. Pose_loss:** pose_loss used by yolov8-nano model

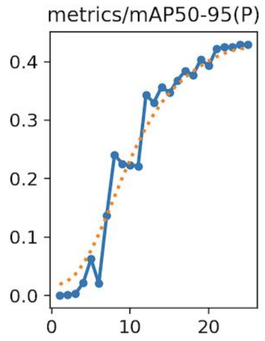
2. **Stage-2:**
   - **2.1. accuracy:** classification accuracy achieved by the model.
   - **2.2. precision:** precision(tp / tp + fp) achieved by the model.
   - **2.3. recall:** recall(tp / tp + fn) achieved by the model.

Results

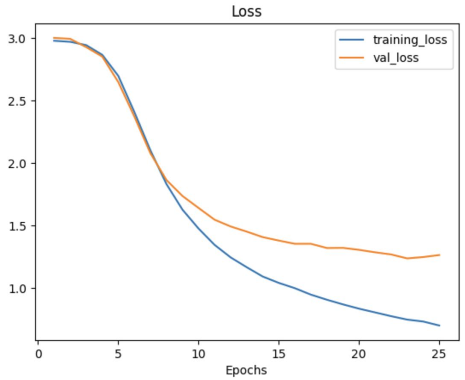
1-> Stage-1:

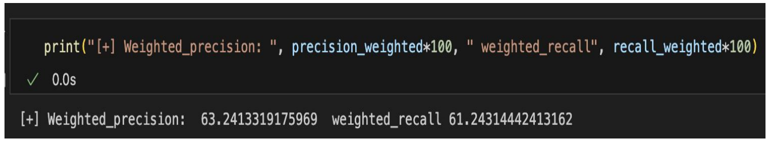




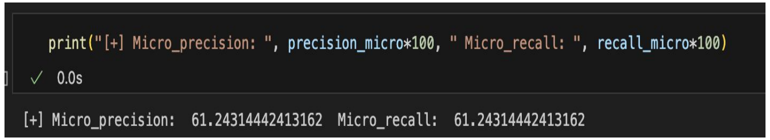


2-> Stage-2:

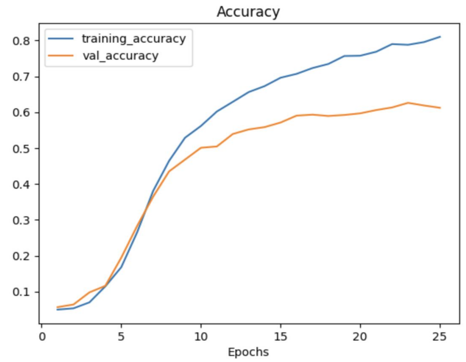




```
print("[+] Weighted_precision: ", precision_weighted*100, " weighted_recall", recall_weighted*100)
✓ 0.0s
[+] Weighted_precision:  63.2413319175969  weighted_recall 61.24314442413162
```



```
print("[+] Micro_precision: ", precision_micro*100, " Micro_recall: ", recall_micro*100)
✓ 0.0s
[+] Micro_precision:  61.24314442413162  Micro_recall:  61.24314442413162
```


Accuracy
training_accuracy
val_accuracy
Epochs