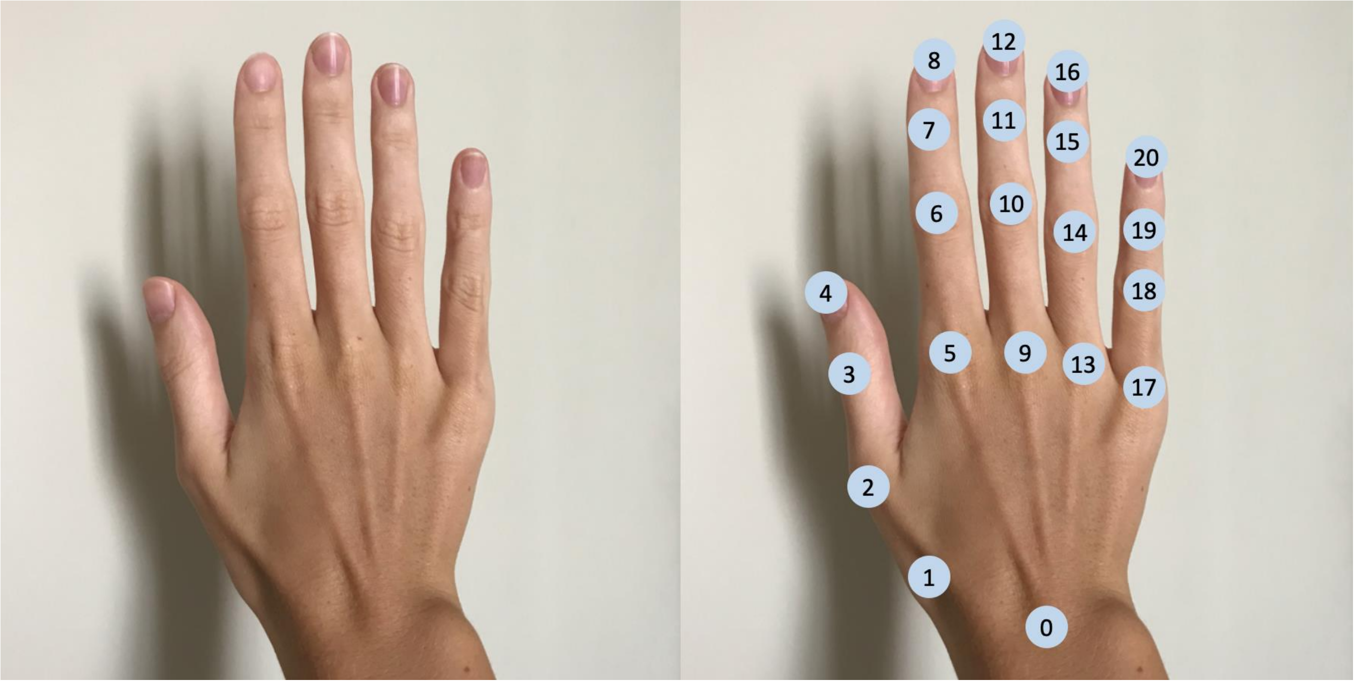


2D HAND POSE ESTIMATION

USING U-NET LIKE ARCHITECTURE

RAMAKRISHNA MISSION

-OISHIK DASGUPTA

-BDA 1st Year

-B2030035

INTRODUCTION

Passive sensing of human hand and limb motion is important for a wide range of applications from human-computer interaction to athletic performance measurement. Human hand pose estimation is a long standing problem in the computer vision and graphics research fields, with a plethora of applications such as:

- machine control,
- augmented and virtual reality.

PROBLEMS

Properties such as:

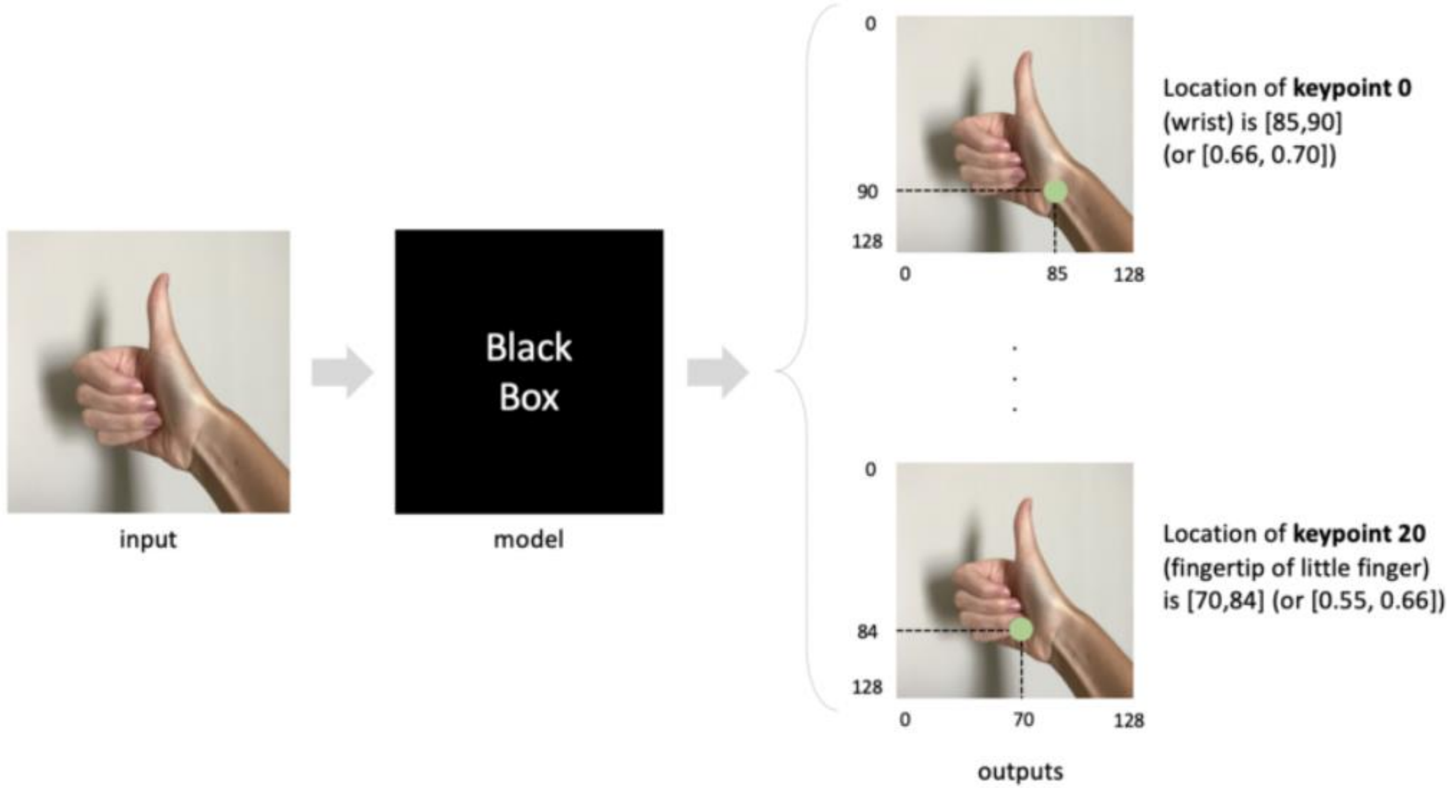
- the hand's morphology,
- occlusions due to interaction with objects,
- appearance diversity due to clothing and jewelry,
- varying lightning conditions and
- different backgrounds,

add extra burden to the nature of the problem.

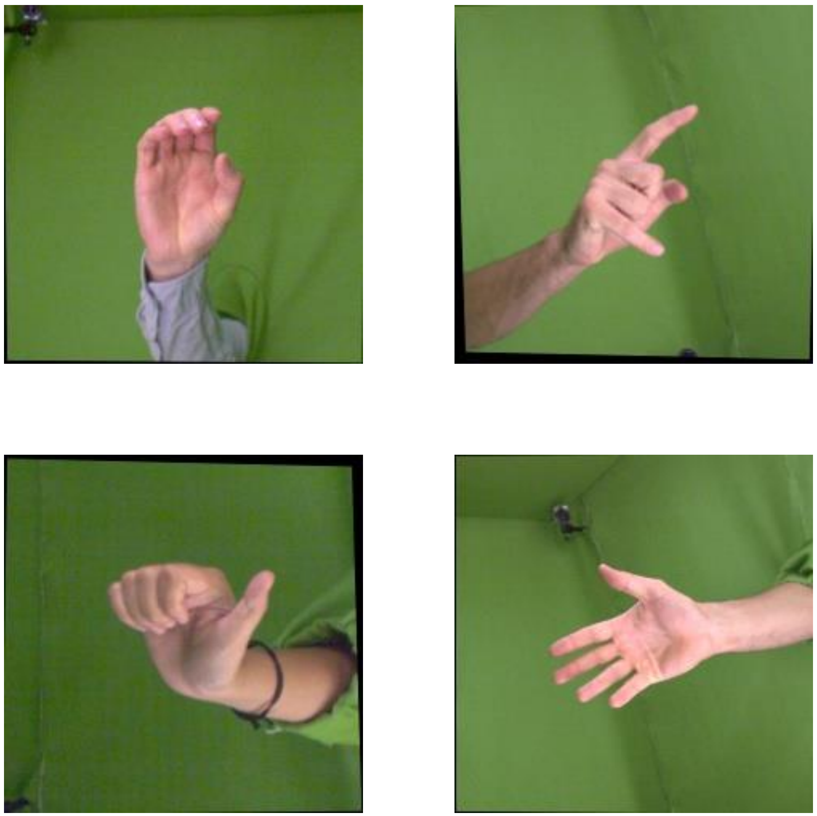
WHAT EXACTLY ARE WE ESTIMATING IN THE HAND POSE ESTIMATION?

WHAT ABOUT THE SECOND HAND?

A TYPICAL 2D HAND POSE ESTIMATOR LOOKS SOMETHING LIKE THIS:



DATASET DESCRIPTION -A



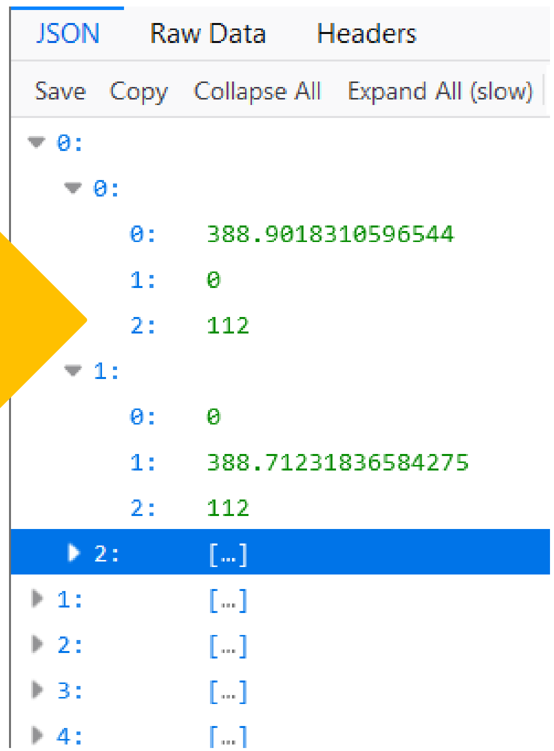
FreiHAND Dataset:
It consists of **130240** images (of the right hand) of dimensions **224*224*3**.

I have worked with the **first 32560 images**. Other images in the dataset are exactly the same as raw ones but with background augmentation.

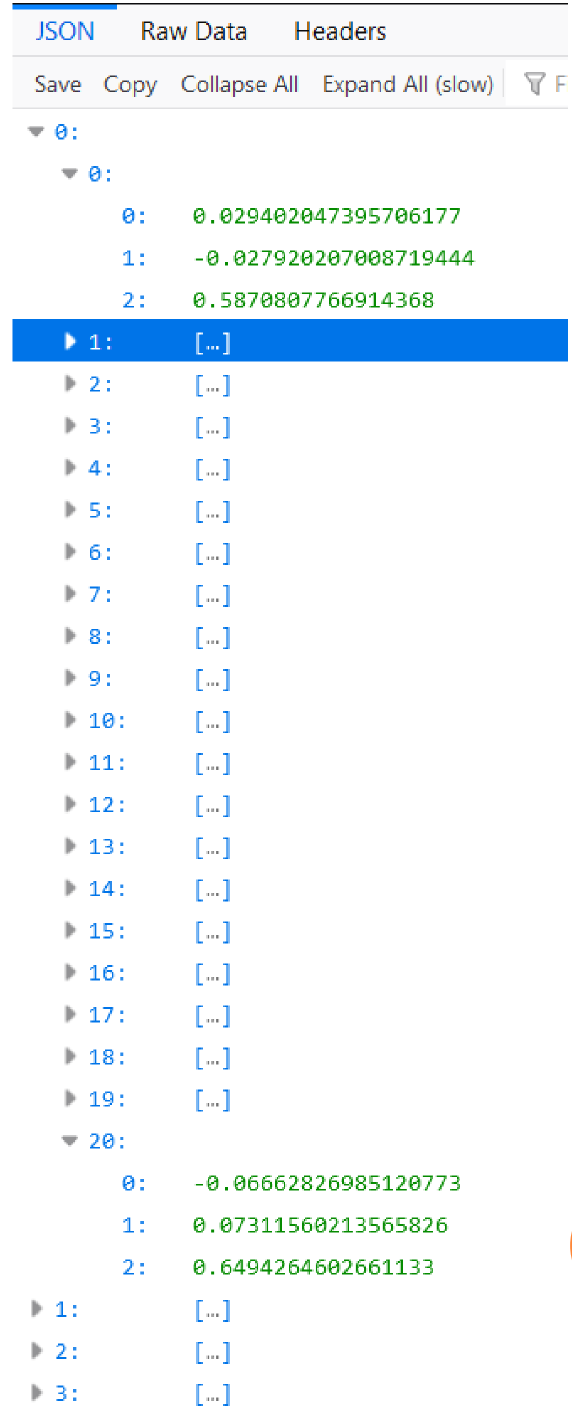
DATASET SPLIT:

Train Set: 0-25999 (80%)
Val. Set: 26000-30999 (15%)
Test Set: 30000-32560 (5%)

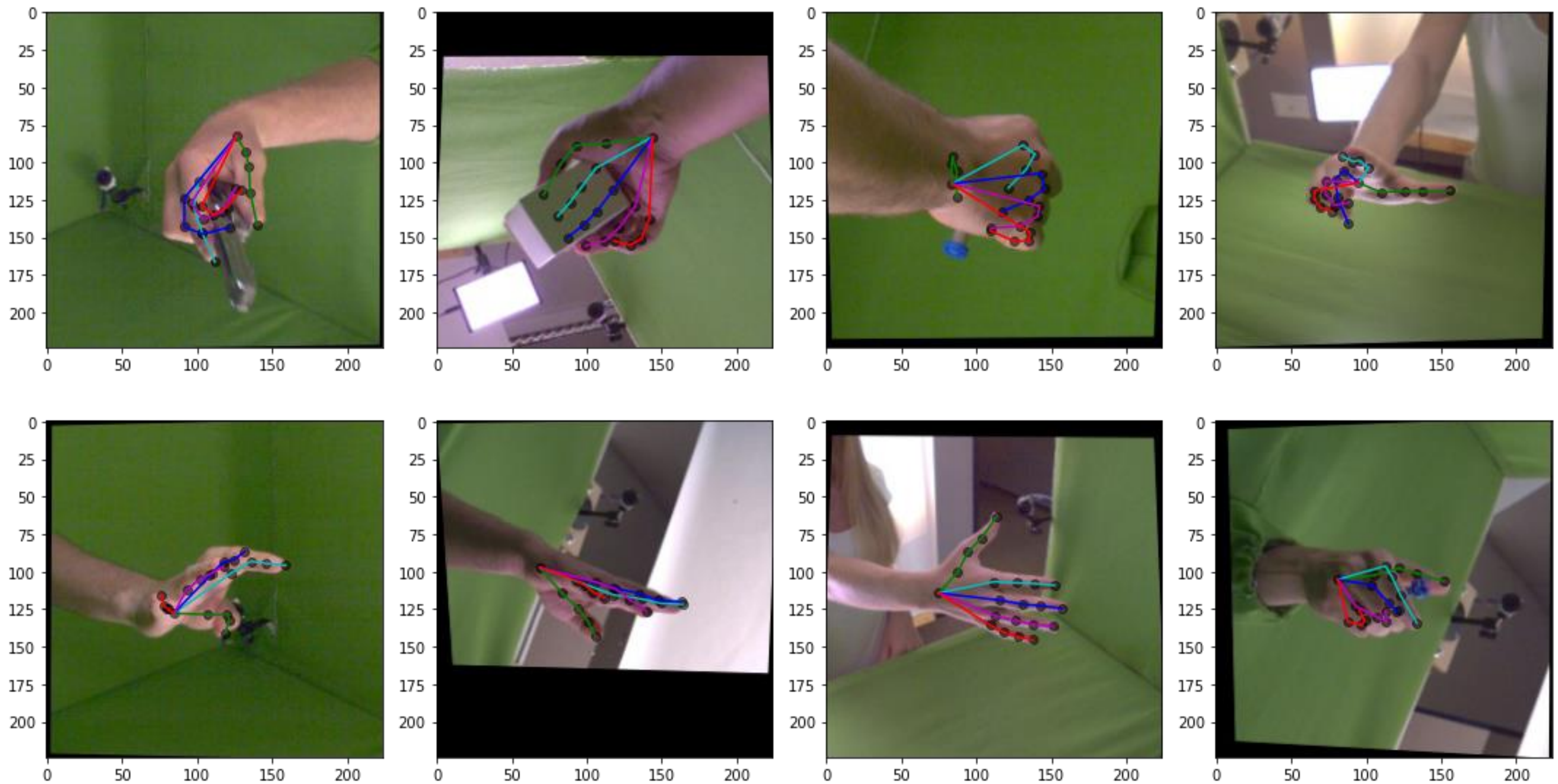
"training_K.json": Array of Camera Intrinsic Matrix for 32560 images



"training_xyz.json": Array of annotations of 21 keypoints for 32560 images

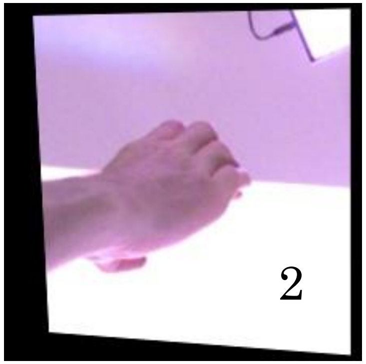


DATASET VISUALIZATION

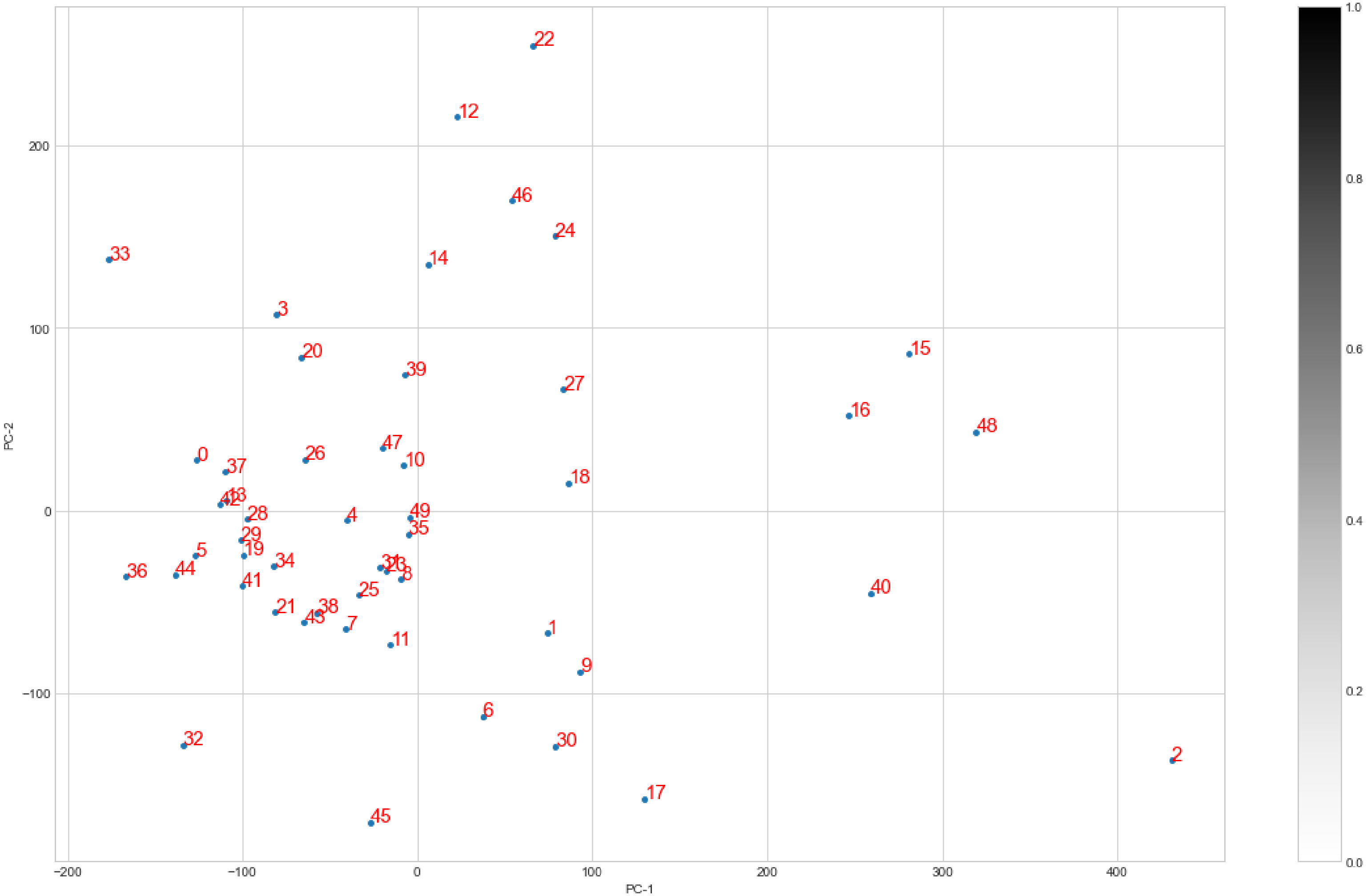


EDA

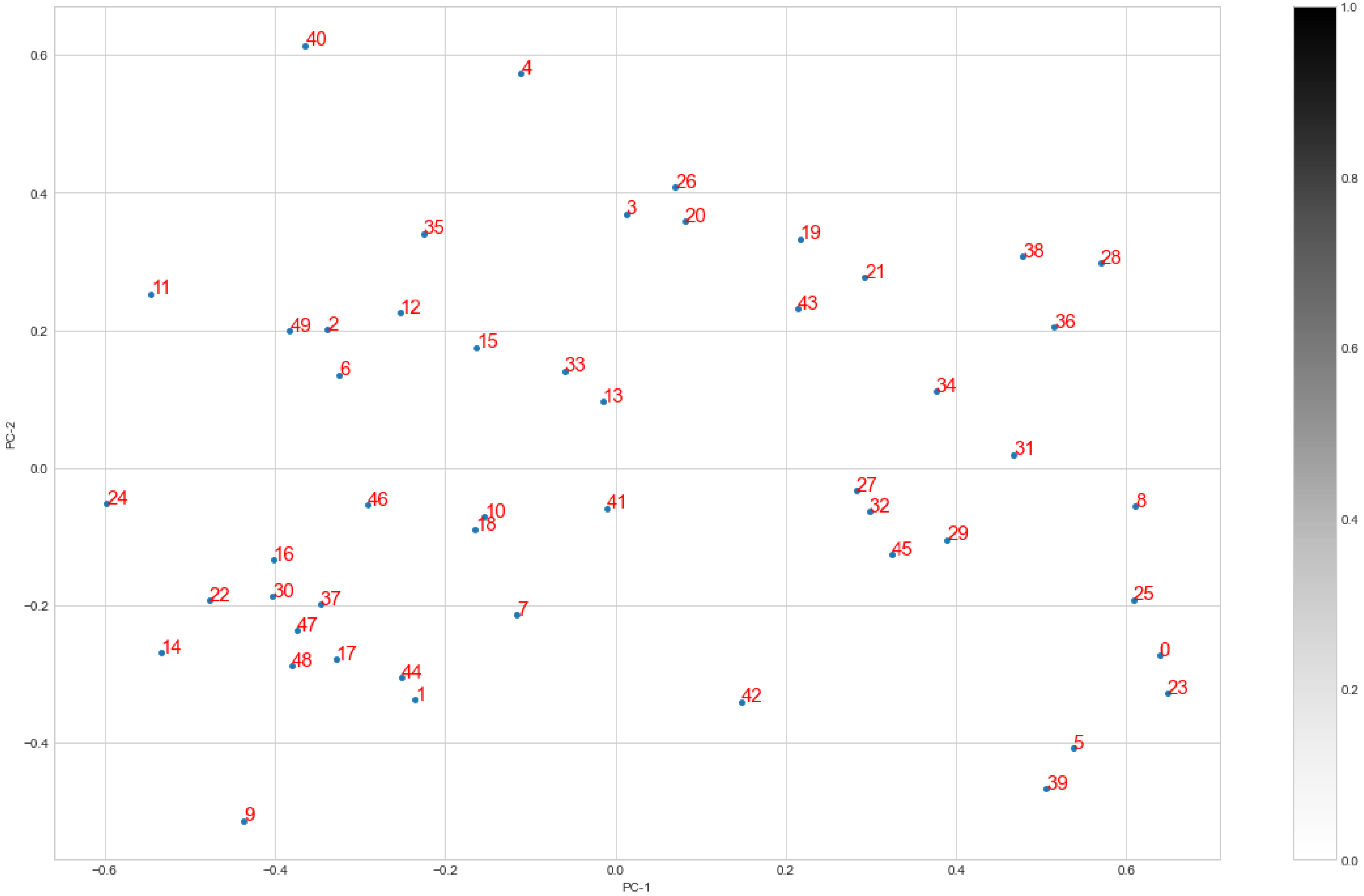
A few sample images



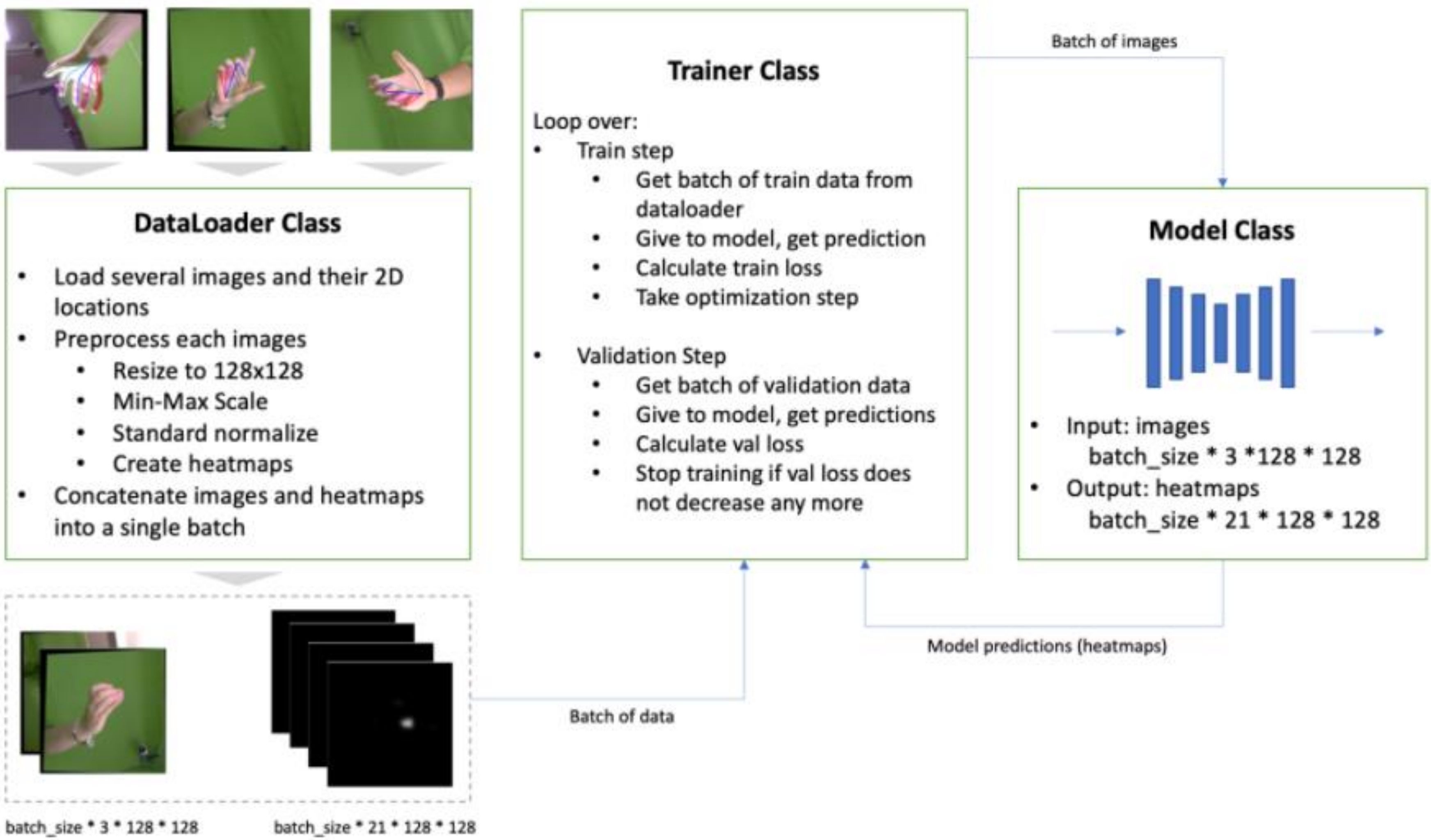
PCA of 50 Sample images from dataset



PCA of 50 Sample images' keypoints from dataset



TRAINING WORKFLOW


Trainer Class
Loop over:
• Train step
• Get batch of train data from dataloader
• Give to model, get prediction
• Calculate train loss
• Take optimization step
• Validation Step
• Get batch of validation data
• Give to model, get predictions
• Calculate val loss
• Stop training if val loss does not decrease any more
Batch of images
Model Class
• Input: images
batch_size * 3 *128 * 128
• Output: heatmaps
batch_size * 21 * 128 * 128
DataLoader Class
• Load several images and their 2D locations
• Preprocess each images
• Resize to 128x128
• Min-Max Scale
• Standard normalize
• Create heatmaps
• Concatenate images and heatmaps into a single batch
Model predictions (heatmaps)
Batch of data
batch_size * 3 * 128 * 128
batch_size * 21 * 128 * 128

Calculation of 2D locations of 21 keypoints from given 3D locations

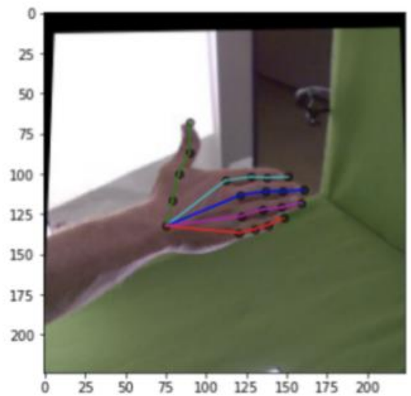
Code Snippet:

```
def projectPoints(xyz, K):
    xyz = np.array(xyz)
    K = np.array(K)
    uv = np.matmul(K, xyz.T).T
    return uv[:, :2] / uv[:, -1:]
```

WHEN RESIZING AN IMAGE, KEYPOINT LOCATIONS SHOULD BE ALSO "RESIZED"

raw

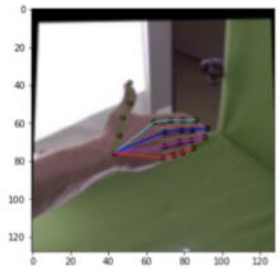
image



keypoint locations

```
array([[ 75.10405108, 132.4615774 ],
       [ 79.24059662, 116.49894932],
       [ 82.93568824,  99.8269972 ],
       [ 89.75880234,  86.83834112],
       [ 89.77472604,  67.69263335],
       [111.98030925, 104.16431663],
       [127.81181736, 101.97436608],
       [138.0194434 , 102.39187217],
       [150.72866321, 101.9796304 ],
       [121.61582695, 113.11492367],
       [136.90265352, 110.9537784 ],
       [147.43718503, 110.91745975],
       [160.80129824, 109.88478528],
       [122.00097724, 126.30052255],
       [135.26404954, 122.08774814],
       [146.3798077 , 121.01411156],
       [158.76500416, 118.00191051],
       [120.24906075, 136.73825998],
       [130.05285117, 134.35906534],
       [138.347327  , 131.89860073],
       [148.08537921, 127.60736257]])
```

resized



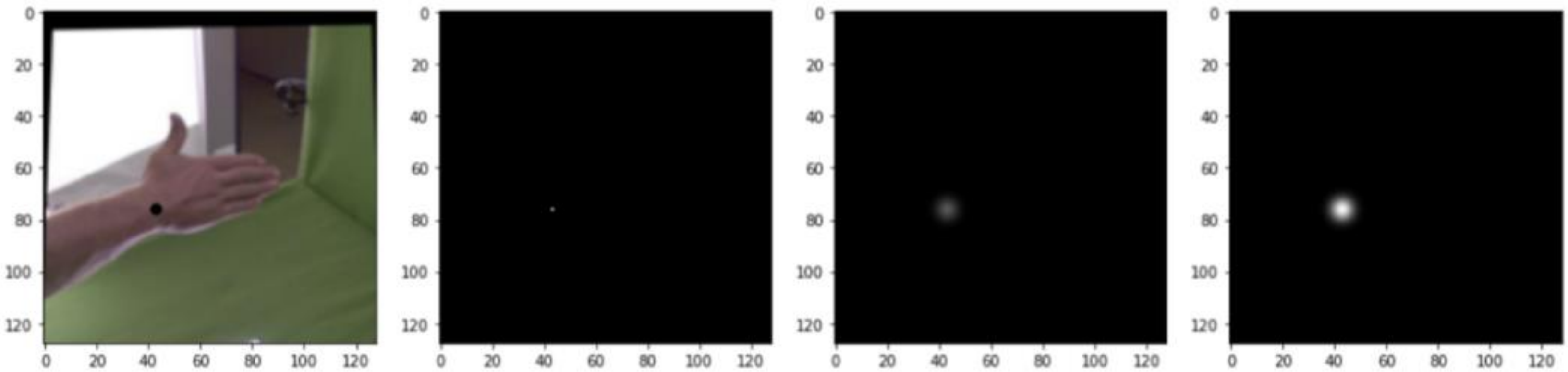
resized_locations = raw_locations / 224 * 128

```
array([[42.91660062, 75.69232994],
       [45.28034092, 66.57082818],
       [47.39182185, 57.0439984 ],
       [51.29074419, 49.62190921],
       [51.29984345, 38.68150477],
       [63.98874814, 59.52246665],
       [73.0353242 , 58.27106633],
       [78.86825337, 58.50964124],
       [86.13066469, 58.27407451],
       [69.49475826, 64.63709924],
       [78.23008773, 63.40215909],
       [84.24982002, 63.38140557],
       [91.88645613, 62.79130588],
       [69.71484414, 72.17172717],
       [77.2937426 , 69.76442751],
       [83.6456044 , 69.15092089],
       [90.72285952, 67.42966315],
       [68.713749  , 78.13614856],
       [74.31591496, 76.77660877],
       [79.05561543, 75.37062899],
       [84.62021669, 72.9184929 ]])
```

CREATE HEATMAPS

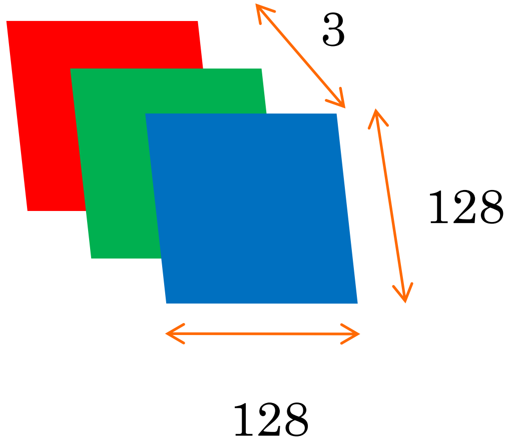
We need to create a separate heatmap for each keypoint, so there will be 21 heatmaps in total.

STEP 1	STEP 2	STEP 3	STEP 4
Choose a keypoint. For instance, a wrist, It's location on resized image is (43,76)	Create array with zeros of size 128x128 Change entry [43,76] to be 1.	Gaussian Blur	Divide array by its max value, so array values are in range [0,1]

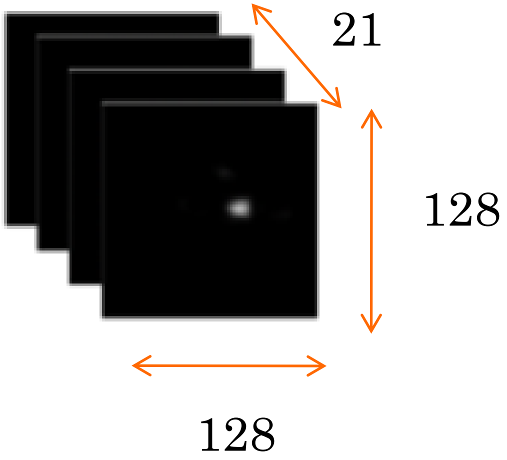


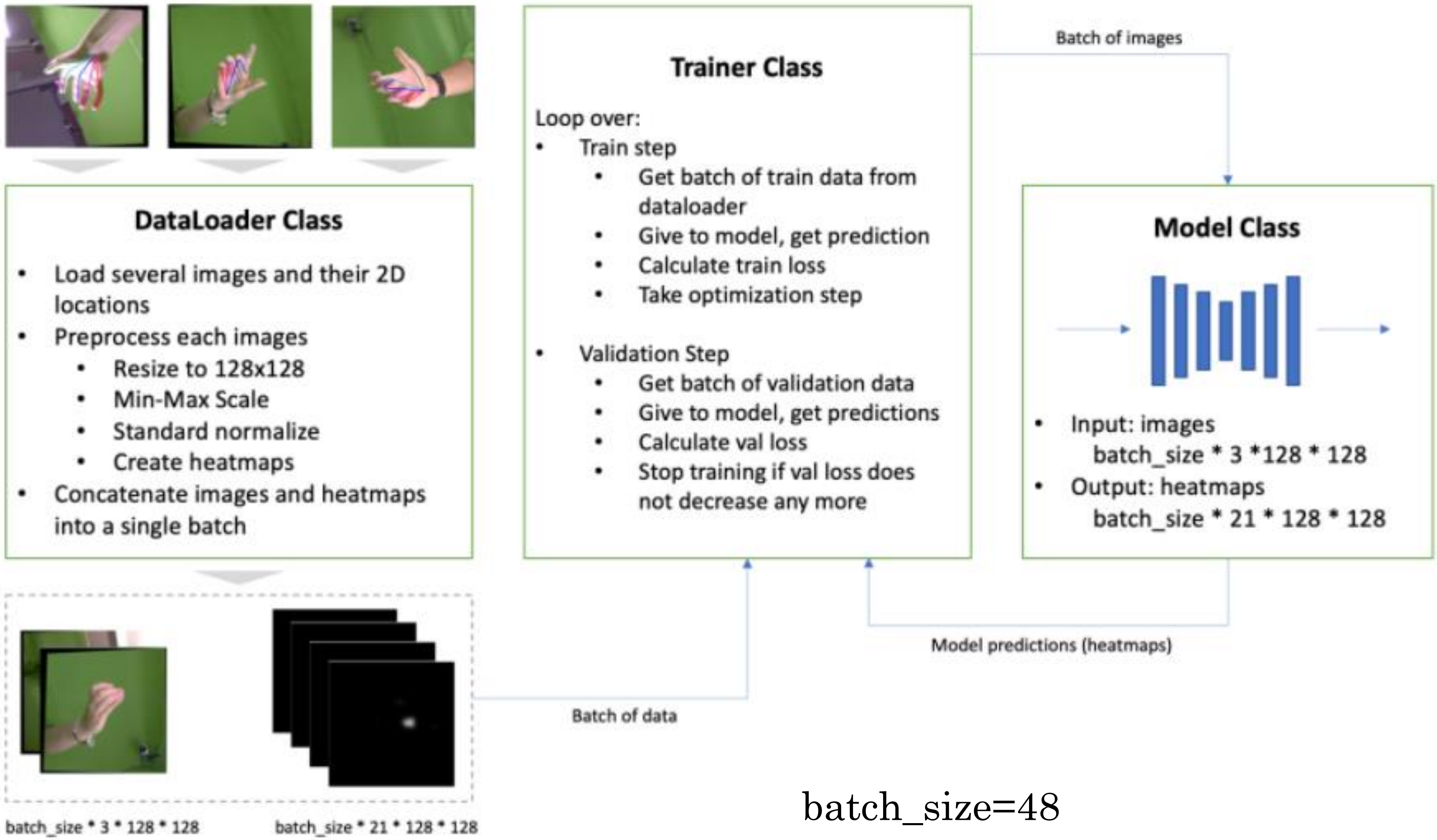
- X is an image of size 3x128x128.
- Y is an array of size 21x128x128, that contains 21 stacked heatmaps.

X:



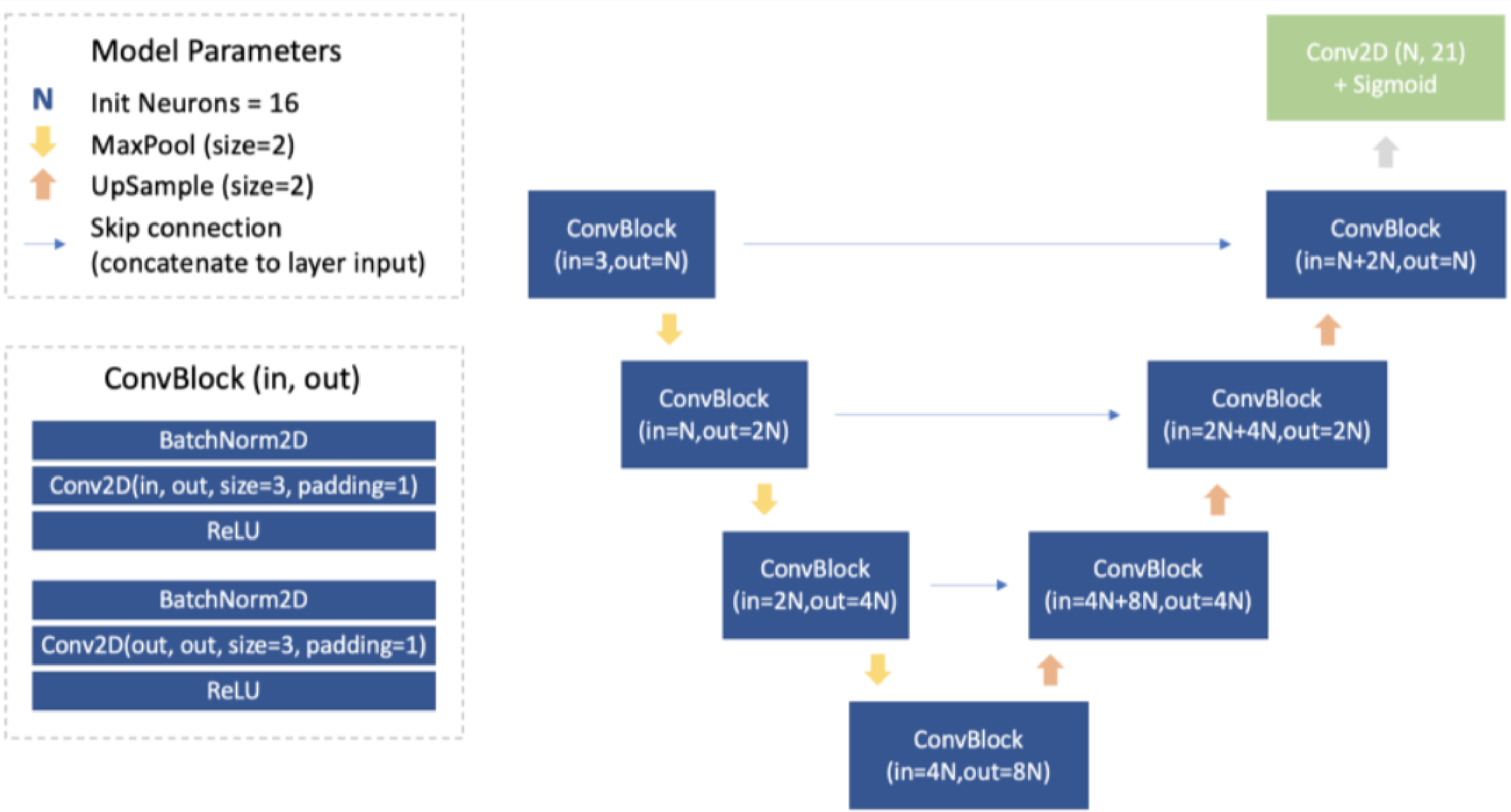
Y:





batch_size=48

MODEL


Model Parameters
N
Init Neurons = 16
MaxPool (size=2)
UpSample (size=2)
Skip connection
(concatenate to layer input)
ConvBlock (in, out)
BatchNorm2D
Conv2D(in, out, size=3, padding=1)
ReLU
BatchNorm2D
Conv2D(out, out, size=3, padding=1)
ReLU
Conv2D (N, 21)
+ Sigmoid
ConvBlock
(in=3,out=N)
ConvBlock
(in=N+2N,out=N)
ConvBlock
(in=N,out=2N)
ConvBlock
(in=2N+4N,out=2N)
ConvBlock
(in=2N,out=4N)
ConvBlock
(in=4N+8N,out=4N)
ConvBlock
(in=4N,out=8N)

LOSS

WHY
IoU LOSS?

LOSS EXPRESSION:

$$\mathbf{I} = \sum_i y_i * t_i$$

$$\mathbf{U} = \sum_i y_i * y_i + \sum_i t_i * t_i - \sum_i y_i * t_i$$

$$\mathbf{IoU} = \frac{\mathbf{I}}{\mathbf{U}}$$

$$\mathbf{L_{IoU}} = \mathbf{1} - \mathbf{IoU}$$

y_i – predicted values
t_i – target values for a pixel in a heatmap.

Loss is calculated for each heatmap seperately, then averaged among the images in the batch.

LOSS CALCULATION:

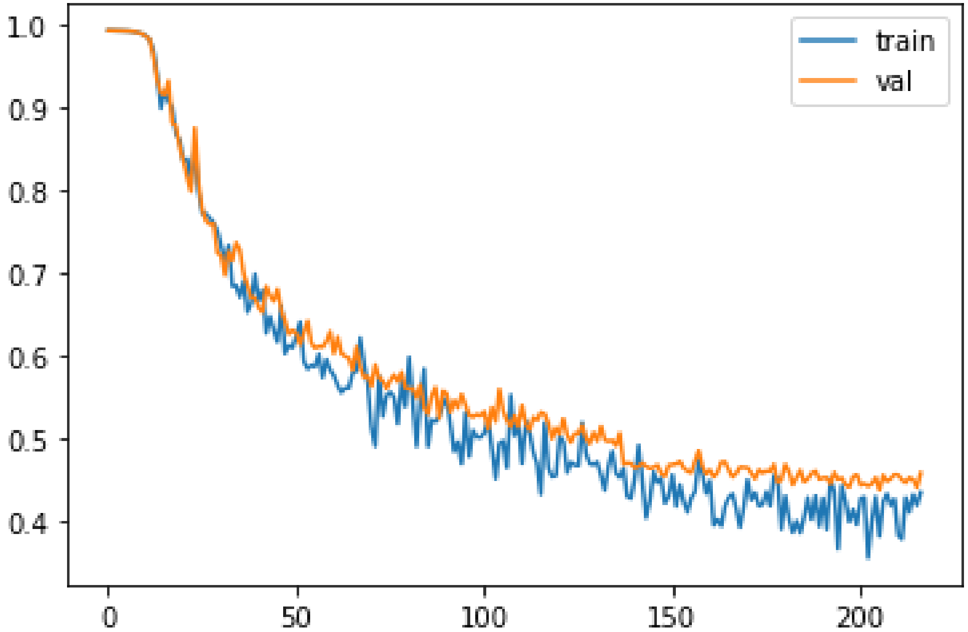
```python
class IoULoss(nn.Module):
    """
    Intersection over Union Loss.
    IoU = Area of Overlap/Area of Union
    IoU loss is modified to use for heatmaps.
    """

    def __init__(self):
        super(IoULoss, self).__init__()
        self.EPSILON = 1e-6

    def _op_sum(self, x):
        return x.sum(-1).sum(-1)

    def forward(self, y_pred, y_true):
        inter = self._op_sum(y_true * y_pred)
        union = (
            self._op_sum(y_true ** 2)
            + self._op_sum(y_pred ** 2)
            - self._op_sum(y_true * y_pred)
        )
        iou = (inter + self.EPSILON) / (union + self.EPSILON)
        iou = torch.mean(iou)
        return 1 - iou
```

TRAINING RESULTS:



Training Loss v/s Validation Loss with progress in epochs

```
Epoch: 215/1000, Train Loss=0.4323002696, Val Loss=0.4503314694
Epoch: 216/1000, Train Loss=0.4195953608, Val Loss=0.4410485086
Epoch: 217/1000, Train Loss=0.4350762367, Val Loss=0.458478973
```

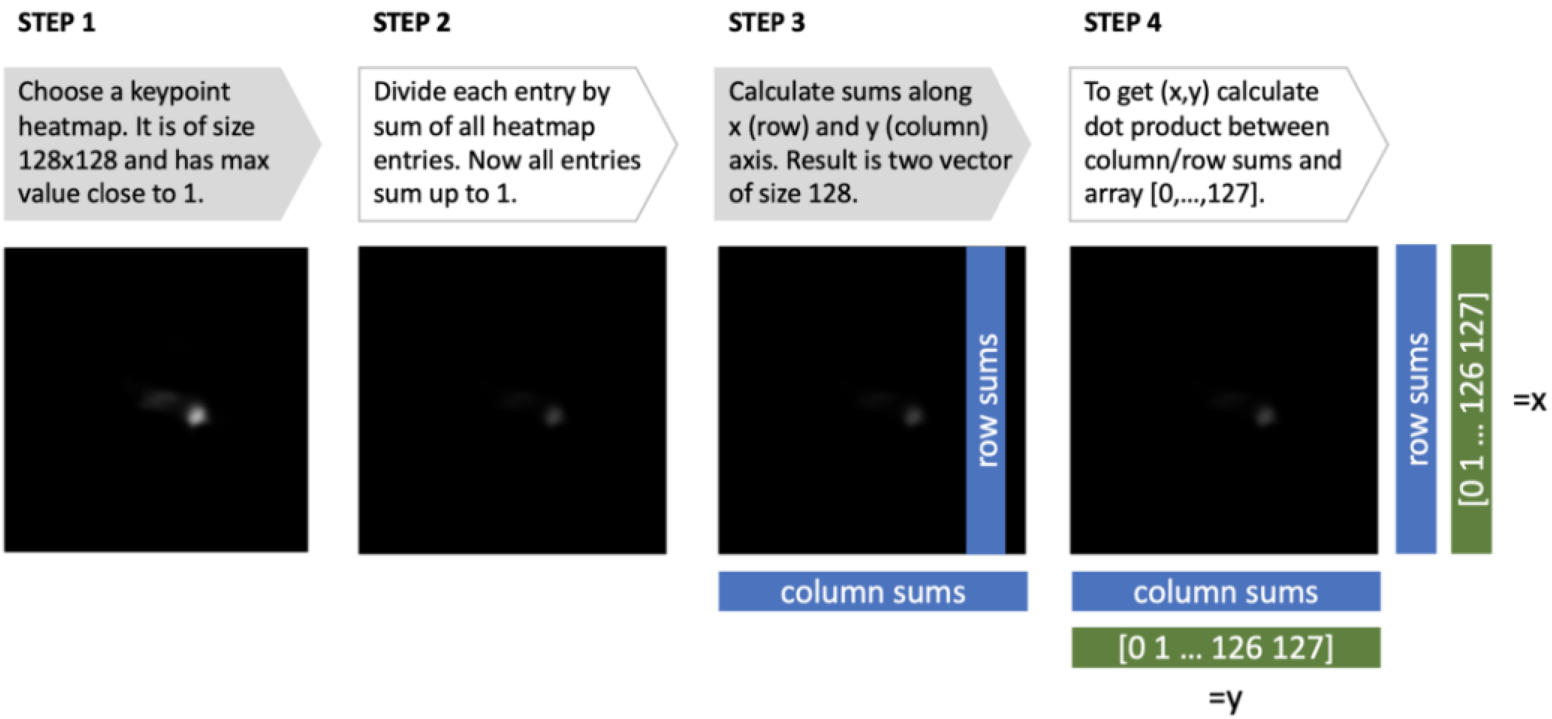
EVALUATION

HEATMAP TO COORDINATES

There are 2 options:

➢Most simply, we may just find a pixel with the largest value in the heatmap. (x,y) location of this pixel is the keypoint location.

➢But a more robust way would be to calculate the average among all heatmap values.



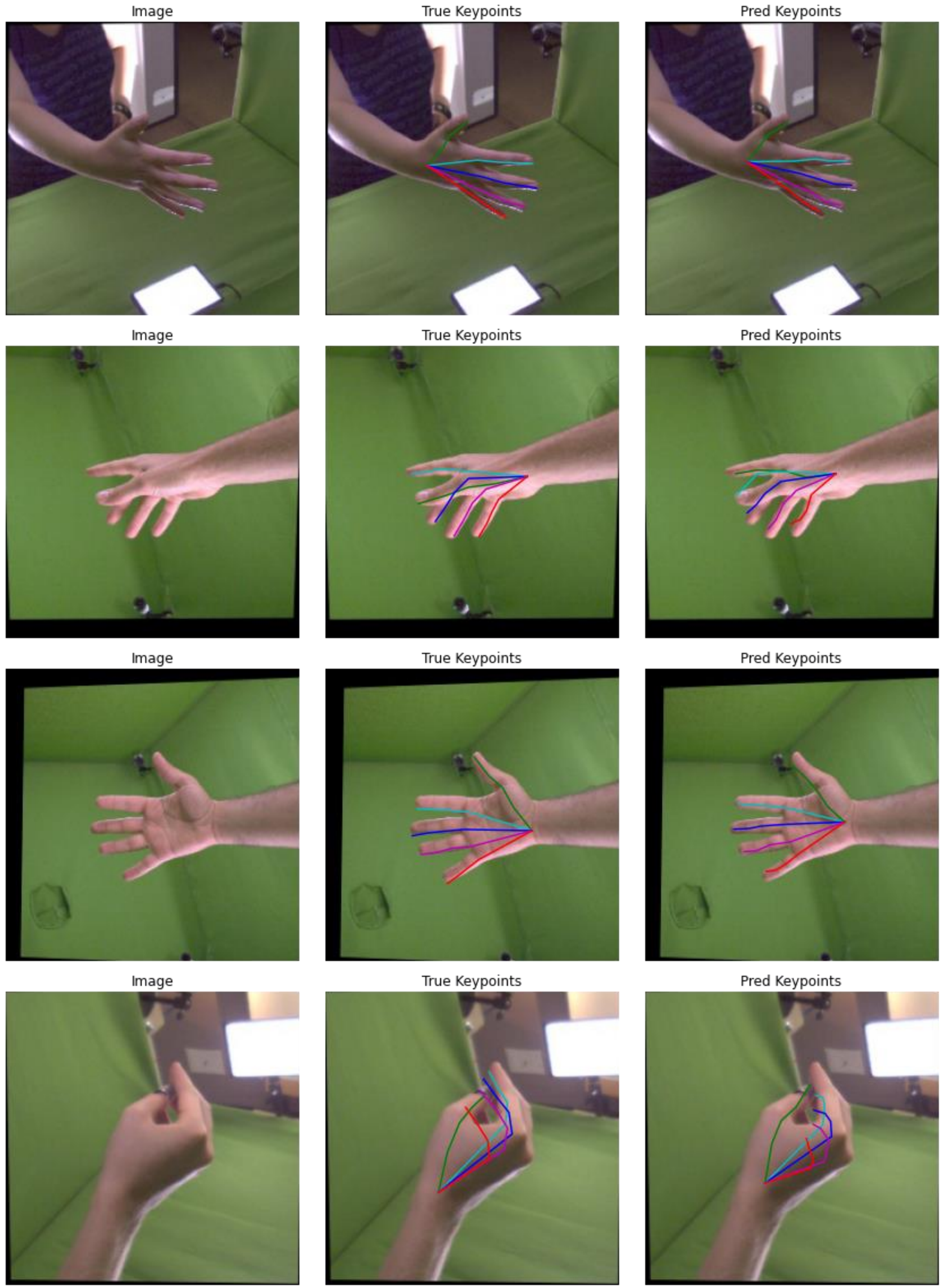


Image
True Keypoints
Pred Keypoints
Image
True Keypoints
Pred Keypoints
Image
True Keypoints
Pred Keypoints
Image
True Keypoints
Pred Keypoints

Evaluation Metric:

- (1) Mean error for each joint:

```
Average error per keypoint: 4.2% from image size
Average error per keypoint: 5 pixels for image 128x128
Average error per keypoint: 10 pixels for image 224x224
```

- (2) Success rate:

  -The proportion of test frames whose average error falls below a threshold

SUCCESS RATE FOR DIFFERENT THRESHOLD VALUES