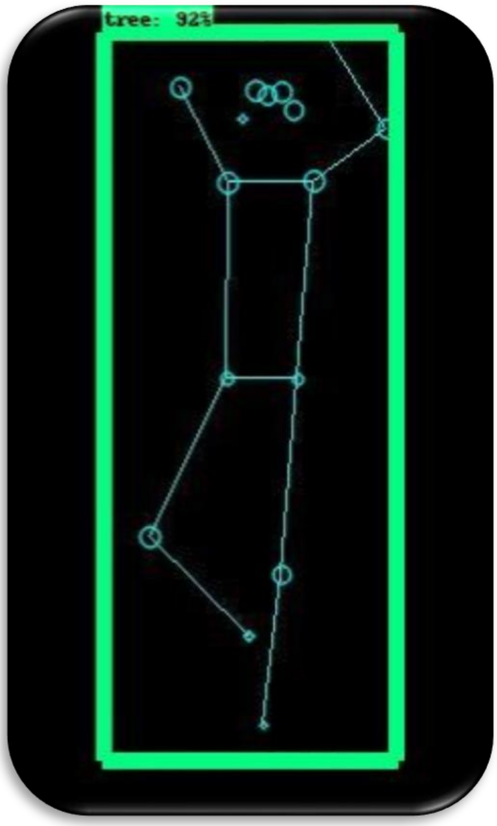
Geometrical Image Processing

Laboratory

Computer Science Faculty

Technion

YOGA MASTER



Barr Assenheimer and Noa Wengrowicz

Supervised by Ron Slossberg

Spring 2020

Abstract:

YOGA MASTER is a platform for yoga practicing anywhere you want. With the Jetson Nano you can take the yoga teacher to the park, to the beach, or just stay at home. Although the teacher is not near you can still get the feedback you need to improve your poses.

Background and motivation:

In the last few years there is an increasing interest in fitness apps and at home training. The main downside of these apps is the lack of real time feedback.
This need was emphasized in the last few months, during the COVID-19 outbreak, forcing people to find alternatives for live fitness classes.

The real-time yoga poses detection on top of the TensorFlow Posenet skeleton tracker can give the trainers the feedback they need to practice from home and improve their yoga poses.

System:

Yoga Master uses the power of AI to make yoga practice from home easier and more efficient. It uses the skeleton created from TensorFlow Posenet and runs it through object detection methods to detect and classify the yoga poses.
The power of the Jetson Nano is reflected exactly in these tasks.

Hardware:

NVIDIA® Jetson Nano™ Developer Kit: a small powerful computer that lets you run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing.

Logitech C160 usb webcam

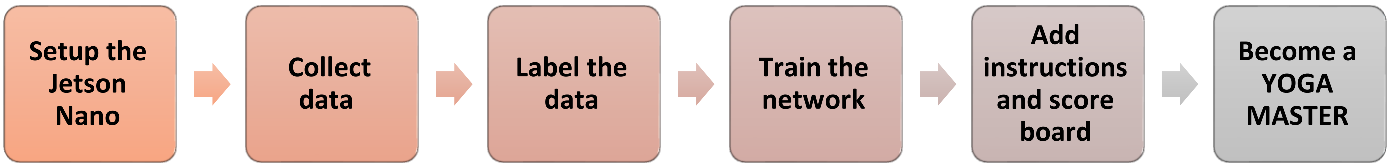
Software:

TensorFlow: an open-source library for numerical computation and large-scale machine- learning.

TensorFlow Posenet: a real-time pose estimation model.

TensorFlow Object Detection API: an open-source framework built on top of TensorFlow for constructing, training and deploying object detection models.

Project's workflow:



Setup the Jetson Nano

Collect data

Label the data

Train the network

Add instructions and score board

Become a YOGA MASTER

Setup the Jetson Nano:

Setting up the Jetson Nano environment was very challenging and took longer than we expected.

We installed a wide range of packages:
Tensorflow, scipy, pyyaml,opencv, libhdf5-serial-dev, hdf5-tools, libhdf5-dev, zlib1g-dev, zip, libjpeg8-dev, numpy, future, mock, h5py, keras_preprocessing, keras_applications, gast, enum34, futures, protobuf, pillow, lxml, Cython, contextlib2, jupyter, matplotlib, pandas, pycocotools, absl-py.

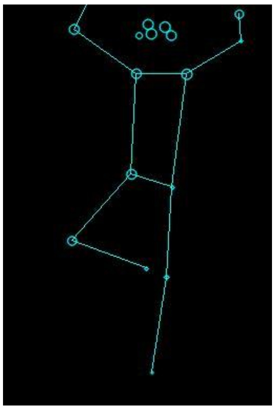
TensorFlow download instructions for Jetson Nano:
https://forums.developer.nvidia.com/t/official-tensorflow-for-jetson-nano/71770

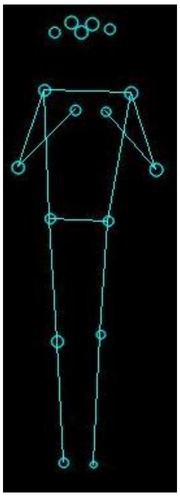
To solve the installations issues we used a virtual environment. Virtual environment allows isolated installs of different Python packages. When you use them, you can have one version of a Python library in one environment and another version in a separate, sequestered environment.

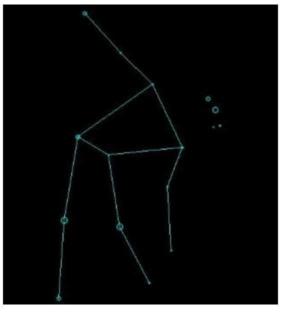
Tip: if you have problems to install packages inside the virtualenv (eg. Permission denied) you can try to install this specific package outside and then create a symbolic link as suggested here- https://stackoverflow.com/questions/56224015/how-to-import-cv2-on-jetson-nano-under-a-virtualenv.
We had this issue while trying to install opencv and the symbolic link fixed it.

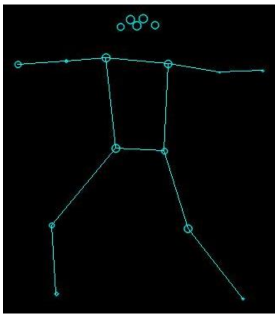
Collect data:

We took 130 pictures for each one of the 4 poses.
We created a variety of images: different people (with different body shapes), different positions in the frame and different angles.



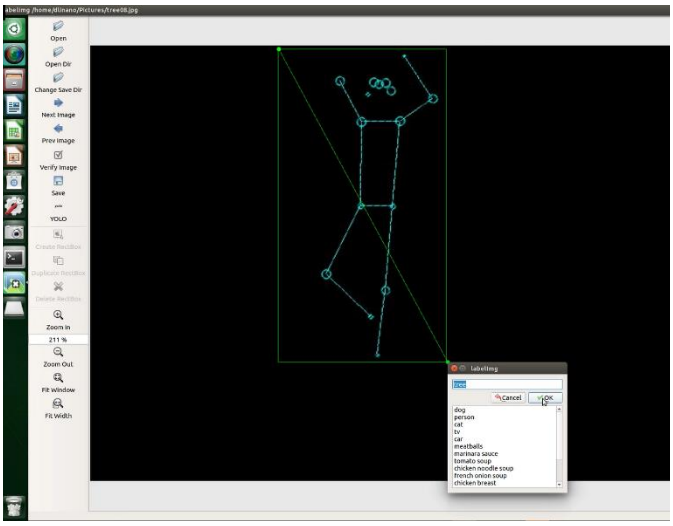






Label the data:

We used LabelImg(https://github.com/tzutalin/labelImg) which is a graphical image annotation tool. Annotations are saved as XML files in PASCAL VOC format.

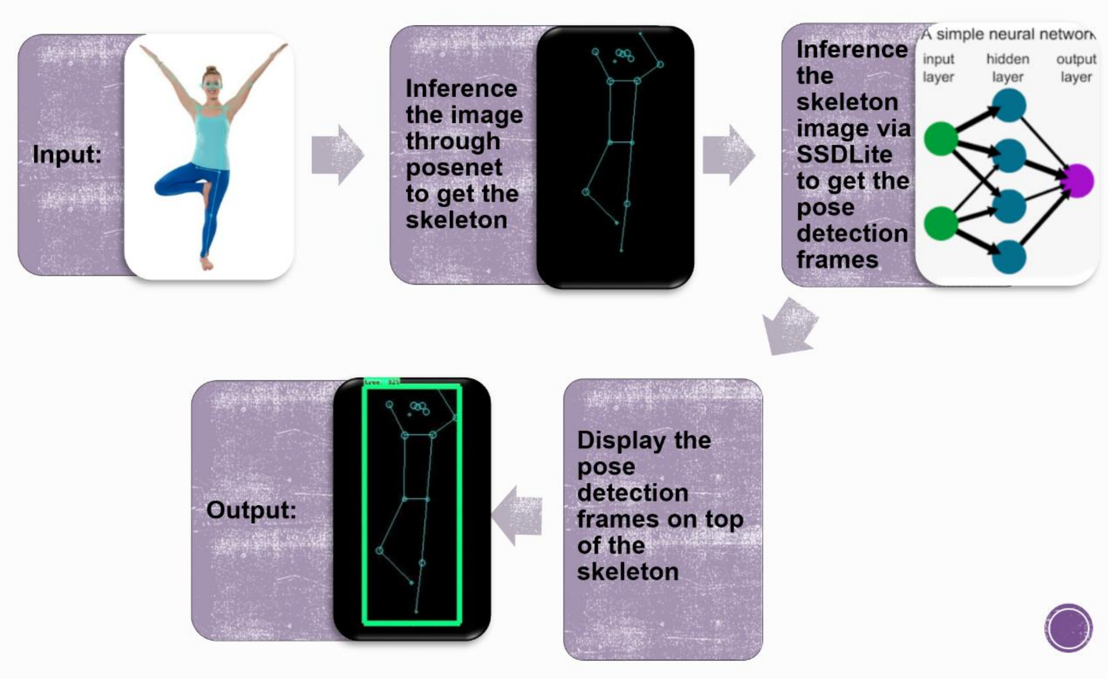


tree

Then we converted the xml files into csv files (separate for train and test) with the script xml_to_csv.py.

Train the network:

We used SSDLite which is a single-Shot multibox Detection (SSD) network intended to perform object detection. This model has an architecture that allows for faster detection but with less accuracy. YOGA MASTER is detecting the poses in real time, so the fast detection is more important than the accuracy.
The input of the network was a skeleton frame we got from Posenet.



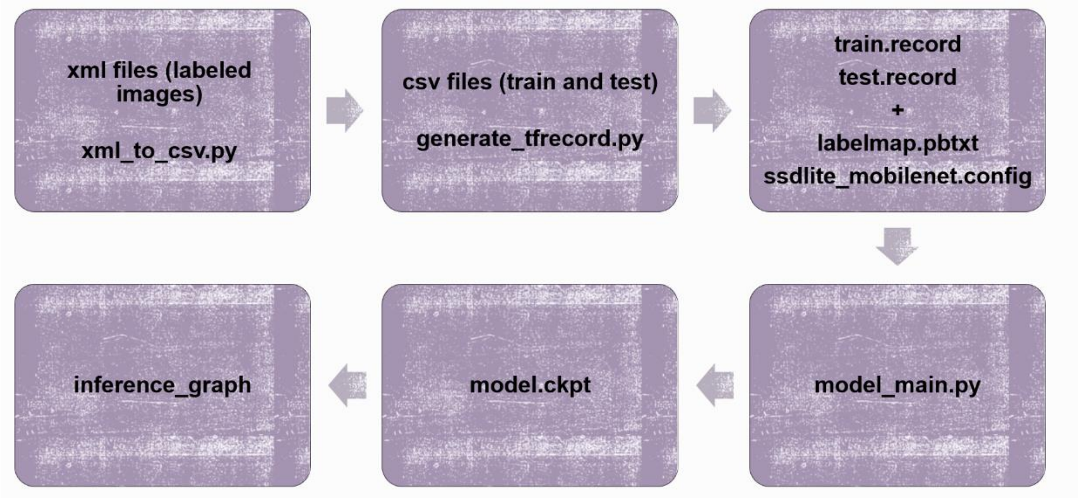
Input:

Inference the image through posenet to get the skeleton

Inference the skeleton image via SSDLite to get the pose detection frames

Display the pose detection frames on top of the skeleton

Output:



xml files (labeled images)

xml_to_csv.py

csv files (train and test)

generate_tfrecord.py

train.record
test.record
+
labelmap.pbtxt
ssdlite_mobilenet.config

model_main.py

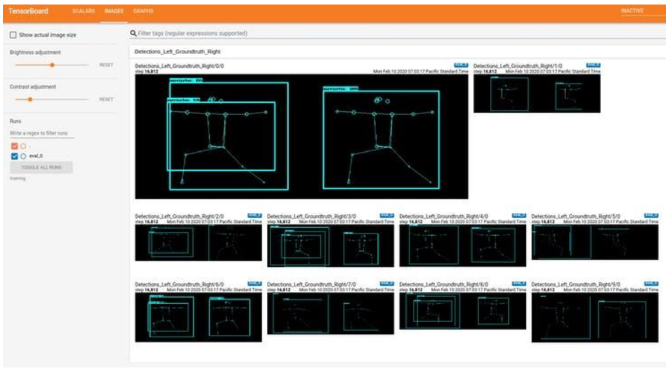
model.ckpt

inference_graph

We tried a few different partitions of the train and test data. In addition, we tried different batch sizes. The best results were obtained when we used 80% of the data for training, and the other 20% for testing, and the batch size was 8.

Training duration-

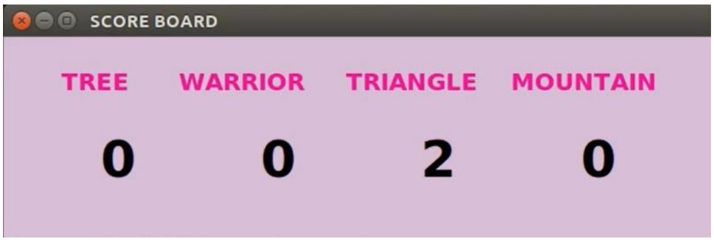
We used TensorBoard to visualize and monitor the training process (model_main.py) and to find the best time to stop the training to avoid overfitting. We ended the training is when the 'DetectionBoxes_Precision mAP' was very close to 1.


TensorBoard
SCALARS
IMAGES
GRAPHS
INACTIVE
□ Show actual image size
Brightness adjustment
RESET
Contrast adjustment
RESET
Runs
Write a regex to filter runs
☑ ○ .
☑ ○ eval_0
TOGGLE ALL RUNS
training
Filter tags (regular expressions supported)
Detections_Left_Groundtruth_Right
Detections_Left_Groundtruth_Right/0/0
step 16,812
Mon Feb 10 2020 07:03:17 Pacific Standard Time
Detections_Left_Groundtruth_Right/1/0
Detections_Left_Groundtruth_Right/2/0
Detections_Left_Groundtruth_Right/3/0
Detections_Left_Groundtruth_Right/4/0
Detections_Left_Groundtruth_Right/5/0
Detections_Left_Groundtruth_Right/6/0
Detections_Left_Groundtruth_Right/7/0
Detections_Left_Groundtruth_Right/8/0
Detections_Left_Groundtruth_Right/9/0

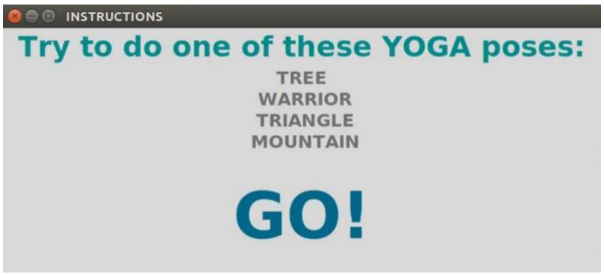
Add instructions and score board:

We used tkinter, the standard Python interface to make the YOGA MASTER GUI.

The scoreboard is a visual histogram that counts how many times the user did each of the poses with accuracy ≥ 80%. The accuracy's percentage is changeable and you can increase it to get harder practice.
The board updates when the user succeeds to do a new pose, different than the one before (it doesn't count the same pose over and over if the user stays in the same pose for a few seconds), that is to make the training more diverse.



TREE	WARRIOR	TRIANGLE	MOUNTAIN
0	0	2	0



Try to do one of these YOGA poses:

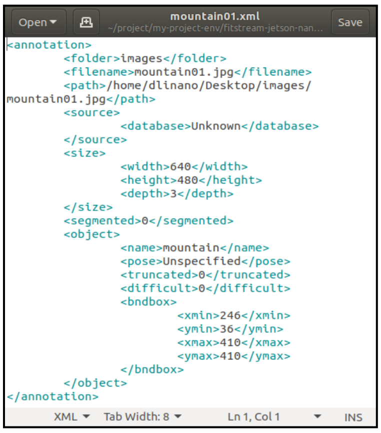
TREE
WARRIOR
TRIANGLE
MOUNTAIN

GO!

Quantitative analysis and results:

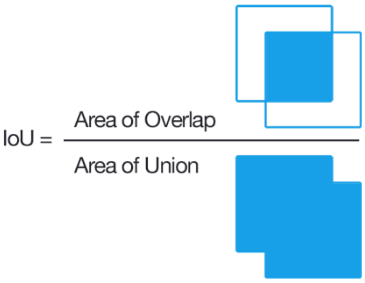
We evaluated the model by IoU - Intersection over Union which is an evaluation metric used to measure the accuracy of an object detector on a particular dataset, by measuring the overlap between two bounding boxes:

1. **The ground-truth bounding boxes**: the hand labeled bounding boxes we created using labelImg. The coordinates if the boxes are stored in xml files which was the output of labelImg tool.



```xml
<annotation>
        <folder>images</folder>
        <filename>mountain01.jpg</filename>
        <path>/home/dlinano/Desktop/images/
mountain01.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>640</width>
                <height>480</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>mountain</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>246</xmin>
                        <ymin>36</ymin>
                        <xmax>410</xmax>
                        <ymax>410</ymax>
                </bndbox>
        </object>
</annotation>
```

2. **The predicted** bounding boxes from our model



The IoU results are in the next couple of pages

Image name	Pose detected	ground truth bounding box				predicted bouning box				IoU	Average
mountain01.jpg	mountain	[36,	246,	410,	410]	[45,	268,	409,	400]	0.781109	0.824819
mountain02.jpg	mountain	[1,	121,	458,	282]	[14,	136,	448,	281]	0.852662	
mountain03.jpg	mountain	[1,	392,	444,	565]	[16,	411,	448,	558]	0.813865	
mountain04.jpg	mountain	[2,	434,	445,	596]	[1,	439,	445,	593]	0.950517	
mountain05.jpg	mountain	[36,	430,	413,	590]	[56,	451,	411,	577]	0.745232	
mountain06.jpg	mountain	[57,	525,	372,	640]	[66,	532,	375,	640]	0.894001	
mountain07.jpg	mountain	[31,	528,	401,	640]	[39,	541,	422,	640]	0.822231	
mountain08.jpg	mountain	[33,	24,	411,	177]	[42,	44,	414,	177]	0.833815	
mountain09.jpg	mountain	[69,	44,	378,	170]	[81,	62,	380,	173]	0.795010	
mountain10.jpg	mountain	[87,	46,	362,	175]	[93,	66,	360,	173]	0.803535	
mountain11.jpg	mountain	[3,	39,	467,	210]	[5,	49,	472,	202]	0.877787	
mountain12.jpg	mountain	[31,	138,	425,	293]	[42,	154,	419,	293]	0.852843	
mountain13.jpg	mountain	[26,	153,	426,	308]	[30,	179,	440,	310]	0.787383	
mountain14.jpg	mountain	[25,	240,	431,	413]	[36,	271,	428,	408]	0.766095	
mountain15.jpg	mountain	[3,	324,	464,	489]	[6,	339,	478,	487]	0.868670	
mountain16.jpg	mountain	[28,	322,	429,	488]	[39,	339,	425,	475]	0.791713	
mountain17.jpg	mountain	[2,	330,	464,	504]	[8,	354,	466,	490]	0.766051	
mountain18.jpg	mountain	[3,	241,	454,	405]	[12,	262,	461,	409]	0.815690	
mountain19.jpg	mountain	[44,	229,	414,	403]	[52,	254,	408,	388]	0.743621	
mountain20.jpg	mountain	[58,	237,	396,	395]	[64,	258,	400,	390]	0.814063	
mountain21.jpg	mountain	[17,	233,	429,	395]	[19,	254,	428,	387]	0.814220	
mountain22.jpg	mountain	[34,	345,	413,	494]	[46,	357,	408,	490]	0.848273	
mountain23.jpg	mountain	[23,	416,	424,	568]	[38,	438,	416,	569]	0.792921	
mountain24.jpg	mountain	[3,	277,	443,	425]	[16,	291,	438,	428]	0.849460	
mountain25.jpg	mountain	[56,	272,	389,	399]	[65,	273,	389,	401]	0.939709	
tree02.jpg	tree	[1,	330,	345,	531]	[7,	356,	346,	536]	0.828195	0.850385
tree03.jpg	tree	[3,	437,	294,	590]	[9,	436,	291,	595]	0.934816	
tree04.jpg	tree	[2,	135,	357,	321]	[19,	156,	364,	316]	0.805396	
tree05.jpg	tree	[12,	192,	331,	353]	[46,	189,	334,	356]	0.852965	
tree07.jpg	tree	[46,	255,	308,	405]	[67,	264,	308,	405]	0.857170	
tree08.jpg	tree	[4,	209,	351,	394]	[13,	236,	357,	382]	0.760911	
tree09.jpg	tree	[1,	106,	349,	291]	[12,	107,	364,	287]	0.901464	
tree10.jpg	tree	[1,	89,	350,	281]	[9,	100,	345,	280]	0.903877	
tree11.jpg	tree	[41,	200,	298,	361]	[58,	205,	313,	371]	0.801161	
tree12.jpg	tree	[1,	211,	364,	446]	[8,	229,	372,	437]	0.849111	
tree13.jpg	tree	[2,	214,	364,	433]	[6,	234,	353,	433]	0.867814	
tree14.jpg	tree	[1,	317,	341,	538]	[7,	345,	346,	534]	0.827500	
tree16.jpg	tree	[18,	388,	283,	563]	[35,	413,	294,	563]	0.762148	
tree17.jpg	tree	[19,	438,	323,	566]	[37,	436,	326,	564]	0.907295	
tree18.jpg	tree	[1,	128,	355,	342]	[3,	152,	357,	343]	0.867233	
tree19.jpg	tree	[1,	226,	342,	423]	[12,	246,	354,	418]	0.812964	
tree20.jpg	tree	[3,	228,	356,	427]	[16,	247,	364,	417]	0.804739	
tree21.jpg	tree	[31,	281,	386,	426]	[54,	303,	386,	425]	0.781668	
tree22.jpg	tree	[55,	265,	359,	400]	[58,	251,	361,	394]	0.856991	
tree23.jpg	tree	[17,	322,	347,	497]	[46,	327,	345,	498]	0.874234	
tree24.jpg	tree	[1,	300,	433,	537]	[1,	306,	440,	539]	0.944941	
tree25.jpg	tree	[1,	345,	431,	603]	[3,	358,	429,	613]	0.905879	
triangle01.jpg	triangle	[149,	219,	382,	467]	[113,	225,	396,	461]	0.790159	0.864016

triangle02.jpg	triangle	[143,	266,	392,	503]	[155,	294,	394,	490]	0.780651	
triangle03.jpg	triangle	[47,	211,	452,	525]	[36,	228,	436,	493]	0.795505	
triangle04.jpg	triangle	[52,	209,	448,	489]	[42,	226,	436,	488]	0.892314	
triangle05.jpg	triangle	[51,	113,	433,	386]	[51,	147,	432,	392]	0.853098	
triangle06.jpg	triangle	[115,	41,	477,	340]	[132,	49,	480,	338]	0.912352	
triangle07.jpg	triangle	[58,	184,	402,	426]	[54,	170,	389,	417]	0.873949	
triangle08.jpg	triangle	[67,	147,	402,	385]	[87,	148,	389,	395]	0.862737	
triangle09.jpg	triangle	[72,	308,	395,	567]	[67,	334,	393,	567]	0.884435	
triangle10.jpg	triangle	[130,	326,	415,	592]	[123,	355,	402,	570]	0.761364	
triangle11.jpg	triangle	[37,	277,	462,	584]	[36,	286,	468,	586]	0.946158	
triangle12.jpg	triangle	[4,	133,	478,	474]	[0,	154,	480,	467]	0.905900	
triangle13.jpg	triangle	[41,	130,	423,	384]	[35,	143,	424,	386]	0.920830	
triangle14.jpg	triangle	[45,	133,	404,	396]	[45,	143,	397,	389]	0.918172	
triangle15.jpg	triangle	[60,	181,	376,	374]	[74,	185,	382,	378]	0.895578	
triangle16.jpg	triangle	[84,	74,	417,	287]	[90,	74,	404,	297]	0.898570	
triangle17.jpg	triangle	[52,	57,	411,	278]	[48,	79,	407,	279]	0.873587	
triangle18.jpg	triangle	[52,	53,	435,	285]	[65,	47,	442,	306]	0.849886	
triangle19.jpg	triangle	[9,	36,	480,	328]	[33,	0,	476,	329]	0.840592	
triangle21.jpg	triangle	[3,	296,	437,	548]	[4,	308,	433,	560]	0.895284	
triangle22.jpg	triangle	[21,	304,	441,	561]	[13,	272,	425,	591]	0.769013	
triangle23.jpg	triangle	[24,	309,	420,	556]	[2,	306,	417,	562]	0.909528	
triangle24.jpg	triangle	[89,	316,	392,	523]	[96,	345,	390,	518]	0.813976	
triangle25.jpg	triangle	[74,	376,	379,	581]	[72,	360,	387,	579]	0.892754	
warrior01.jpg	warrior	[20,	190,	353,	487]	[34,	233,	367,	487]	0.788213	
warrior02.jpg	warrior	[27,	185,	356,	510]	[31,	216,	353,	495]	0.840566	
warrior03.jpg	warrior	[18,	167,	364,	526]	[18,	201,	361,	502]	0.831171	
warrior04.jpg	warrior	[22,	211,	359,	563]	[32,	247,	357,	554]	0.842306	
warrior05.jpg	warrior	[4,	219,	381,	601]	[7,	254,	392,	581]	0.826129	
warrior06.jpg	warrior	[1,	83,	392,	444]	[2,	94,	399,	442]	0.941017	
warrior07.jpg	warrior	[4,	220,	400,	578]	[12,	236,	396,	574]	0.917601	
warrior08.jpg	warrior	[27,	99,	370,	444]	[35,	131,	367,	425]	0.827368	
warrior09.jpg	warrior	[42,	126,	371,	455]	[43,	144,	370,	436]	0.884023	
warrior10.jpg	warrior	[75,	139,	351,	402]	[88,	152,	350,	397]	0.884804	
warrior11.jpg	warrior	[69,	127,	347,	432]	[72,	144,	344,	417]	0.873912	
warrior12.jpg	warrior	[73,	230,	341,	504]	[93,	280,	341,	495]	0.722566	
warrior13.jpg	warrior	[87,	244,	352,	491]	[88,	276,	350,	481]	0.820775	0.851272
warrior14.jpg	warrior	[68,	211,	350,	497]	[85,	235,	351,	480]	0.803119	
warrior15.jpg	warrior	[2,	312,	376,	627]	[23,	337,	385,	631]	0.833746	
warrior16.jpg	warrior	[11,	328,	375,	623]	[34,	356,	375,	624]	0.840885	
warrior17.jpg	warrior	[2,	358,	400,	640]	[3,	378,	401,	632]	0.894800	
warrior18.jpg	warrior	[43,	328,	350,	587]	[53,	335,	338,	594]	0.880559	
warrior19.jpg	warrior	[44,	286,	351,	585]	[59,	323,	339,	573]	0.759129	
warrior20.jpg	warrior	[1,	31,	390,	398]	[14,	56,	390,	391]	0.877324	
warrior21.jpg	warrior	[78,	126,	382,	359]	[74,	143,	371,	356]	0.875144	
warrior22.jpg	warrior	[81,	100,	382,	353]	[85,	118,	390,	330]	0.810584	
warrior23.jpg	warrior	[60,	52,	433,	396]	[57,	61,	426,	397]	0.945232	
warrior24.jpg	warrior	[44,	42,	455,	363]	[46,	60,	444,	362]	0.912111	
warrior25.jpg	warrior	[89,	137,	381,	328]	[94,	147,	366,	321]	0.848707	
0.847366											

Results:

Average IoU of the total test set is 84.73%
Average IoU of the 'Tree' pose test image set is 85.03%
Average IoU of the 'Triangle' pose test image set is 86.4%
Average IoU of the 'Mountain' pose test image set is 82.48%
Average IoU of the 'Warrior' pose test image set is 85.12%

Conclusions:

- The video streaming gives better result when running YOGA MATER directly on the Jetson Nano and not via SSH .
- The Jetson Nano environment is challenging and setting it up takes long time.
- Virtual environments are very effective and easy to use.
- USB camera worked better for us.
- Training the model on the Jetson Nano is too slow. It is better to use a powerful GPU.
- Best training results were obtained when :
  - The batch size was 8.
  - We ended the training is when the ‘DetectionBoxes_Precision mAP’ is very close to 1.
  - We used 80% of the data for training, and the other 20% for testing.

Future work:

- Add more yoga poses.
- Add different fitness activities.
- Augment existing images.
- Improve the network by adding coordinates vector to the data set.

Sources:

- https://www.hackster.io/mixpose/mixpose-722df5#toc-step3--tensorflow-and-training-pose-data-5
- https://github.com/MixPose/MixPose-Jetson-Nano
- https://github.com/tensorflow/models/tree/master/research/object_detection