[26/03/22] Learning Note EfficientPose & BlazePose

About EfficientPose

https://arxiv.org/pdf/2004.12186.pdf

-Developed based on EfficientNet (more computationally efficient architecture), and an improvement of OpenPose.

-Publicly accessible scalable single-person pose estimation, provide a simple intuitive interface for **high-precision movement extraction** from 2D images, videos, or directly from your web camera.

-Previous solution: OpenPose.

- Drawback: the level of detail in keypoint estimates is limited due to its low-resolution outputs ⇒ less suitable for **precision-demanding applications**, such as elite sports and medical assessments (depend on high degree of precision in the assessment of movement kinematics).

-Architecture:

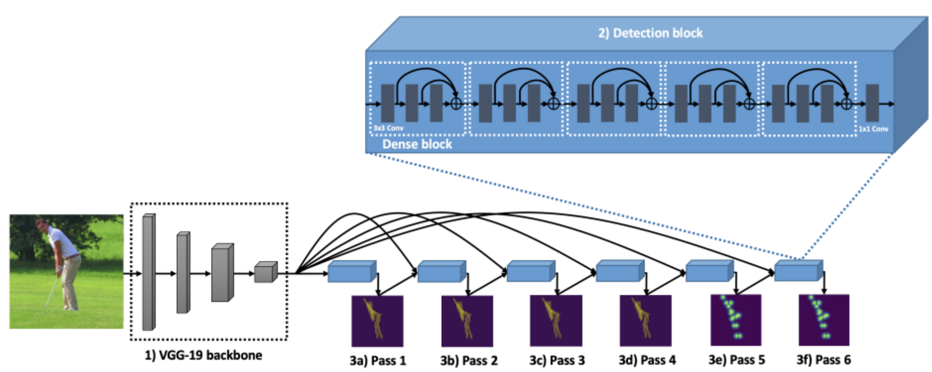


Fig. 1 OpenPose architecture utilizing 1) VGG-19 feature extractor, and 2) 4+2 passes of detection blocks performing 4+2 passes of estimating part affinity fields (3a-d) and confidence maps (3e and 3f)

OpenPose architecture

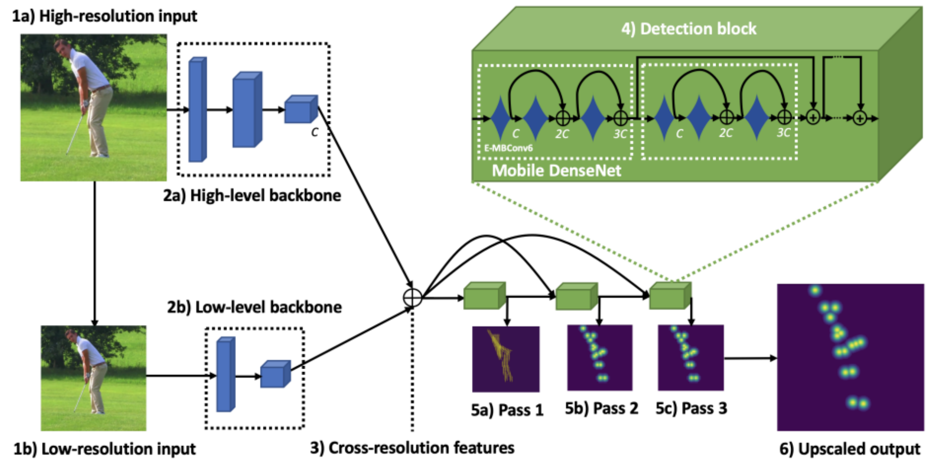


Fig. 2 Proposed architecture comprising 1a) high-resolution and 1b) low-resolution inputs, 2a) high-level and 2b) low-level EfficientNet backbones combined into 3) cross-resolution features, 4) Mobile DenseNet detection blocks, 1+2 passes for estimation of part affinity fields (5a) and confidence maps (5b and 5c), and 6) bilinear upscaling

EfficientPose architecture

- Differences in architecture of EfficientPose compared to OpenPose:
  - EfficientPose utilizes both high and low-resolution input images.
  - EfficientPose have scalable EfficientNet backbones.
  - Cross-resolution features.
  - Scalable Mobile DenseNet detection blocks in fewer detection passes.
  - Bilinear upscaling.
- **Figure 2 step 2a) and 2b)** Feature extractor of EfficientPose based on initial blocks of EfficientNet, pretrained on ImageNet.

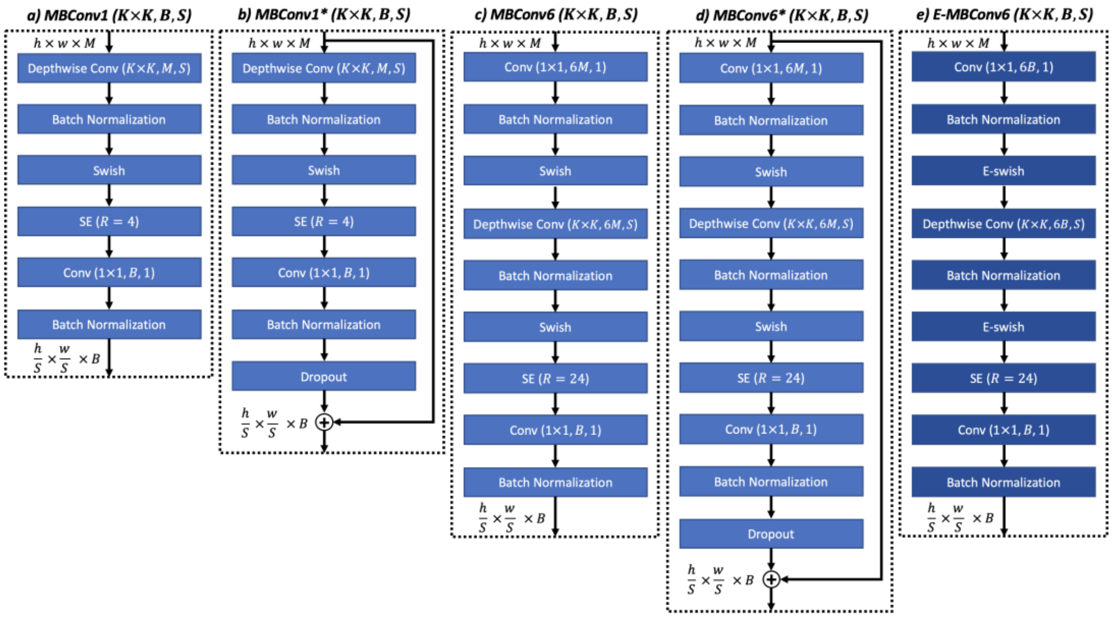


Fig. 3 The composition of MBConvs. From left: a-d) $MBConv(K \times K, B, S)$ in EfficientNets performs depthwise convolution with filter size $K \times K$ and stride S, and outputs B feature maps. $MBConv^*$ (b and d) extends regular MBConvs by including dropout layer and skip connection. e) $E\text{-}MBConv6(K \times K, B, S)$ in Mobile DenseNets adjusts $MBConv6$ with E-swish activation and number of feature maps in expansion phase as $6B$. All MBConvs take as input M feature maps with spatial height and width of h and w, respectively. R is the reduction ratio of SE

-Summary: The EfficientPose framework comprises a family of five ConvNets (i.e., EfficientPose I-IV and RT) that are constructed by compound scaling.

- EfficientPose **exploits the advances in computationally efficient** ConvNets for image recognition to construct a **scalable network architecture** that is **capable of performing single-person HPE** across different computational constraints. ⇒ chậm hơn :?
- It utilizes **both high and low-resolution images** to provide two separate viewpoints that are processed independently through high and low-level backbones.
- ⇒ Resulting features are concatenated to produce **cross-resolution features**, enabling selective emphasis on **global** and **local image information**.

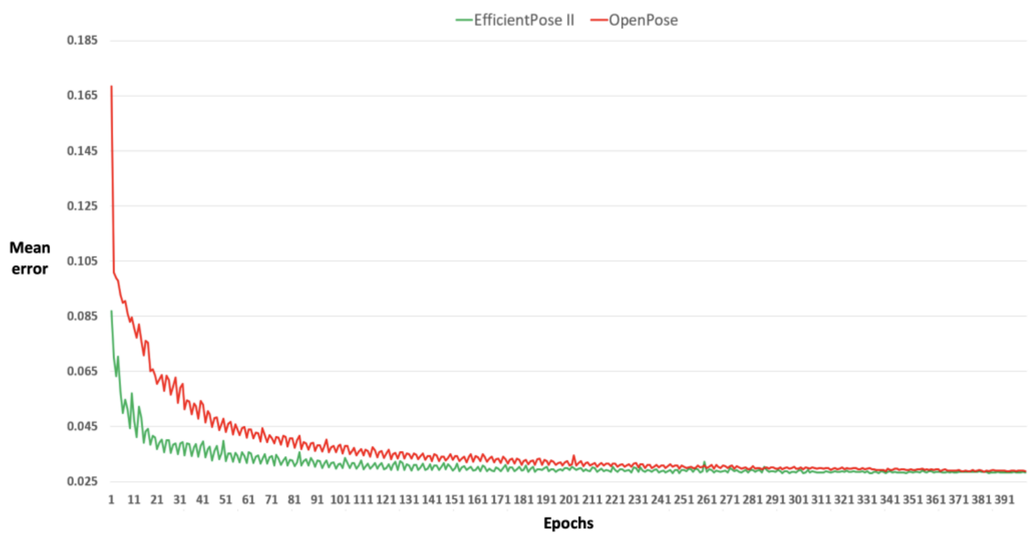
-Compare result with OpenPose:

Table 3 Performance of EfficientPose compared to OpenPose on the MPII validation dataset, as evaluated by efficiency (number of parameters and FLOPs, and relative reduction in parameters and FLOPs compared to OpenPose) and accuracy (mean PCK_h@50 and mean PCK_h@10)

Model	Parameters	Parameter reduction	FLOPs	FLOP reduction	PCK_h@50	PCK_h@10
OpenPose [6]	$25.94M$	$1\times$	$160.36G$	$1\times$	87.60	22.76
EfficientPose RT	$0.46M$	$56\times$	$0.87G$	$184\times$	82.88	23.56
EfficientPose I	$0.72M$	$36\times$	$1.67G$	$96\times$	85.18	26.49
EfficientPose II	$1.73M$	$15\times$	$7.70G$	$21\times$	88.18	30.17
EfficientPose III	$3.23M$	$8.0\times$	$23.35G$	$6.9\times$	89.51	30.90
EfficientPose IV	$6.56M$	$4.0\times$	$72.89G$	$2.2\times$	89.75	35.63

Table 4 State-of-the-art results in PCK_h@50 (both for individual body parts and overall mean value) on the official MPII test dataset [1] compared to the number of parameters

Model	Parameters	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Mean
Pishchulin et al., ICCV'13 [35]	–	74.3	49.0	40.8	32.1	36.5	34.4	35.2	44.1
Tompson et al., NIPS'14 [53]	–	95.8	90.3	80.5	74.3	77.6	69.7	62.8	79.6
Lifshitz et al., ECCV'16 [28]	$76M$	97.8	93.3	85.7	80.4	85.3	76.6	70.2	85.0
Tang et al., BMVC'18 [50]	$10M$	97.4	96.2	91.8	87.3	90.0	87.0	83.3	90.8
Newell et al., ECCV'16 [33]	$26M$	98.2	96.3	91.2	87.1	90.1	87.4	83.6	90.9
Zhang et al., CVPR'19 [60]	$3M$	98.3	96.4	91.5	87.4	90.9	87.1	83.7	91.1
Bulat et al., FG'20 [5]	$9M$	98.5	96.4	91.5	87.2	90.7	86.9	83.6	91.1
Yang et al., ICCV'17 [57]	$27M$	98.5	96.7	92.5	88.7	91.1	88.6	86.0	92.0
Tang et al., ECCV'18 [49]	$16M$	98.4	96.9	92.6	88.7	91.8	89.4	86.2	92.3
Sun et al., CVPR'19 [44]	$29M$	98.6	96.9	92.8	89.0	91.5	89.0	85.7	92.3
Zhang et al., arXiv'19 [61]	$24M$	98.6	97.0	92.8	88.8	91.7	89.8	86.6	92.5
OpenPose [6]	$25.94M$	97.7	94.7	89.5	84.7	88.4	83.6	79.3	88.8
EfficientPose RT	$0.46M$	97.0	93.3	85.0	79.2	85.9	77.0	71.0	84.8
EfficientPose IV	$6.56M$	98.2	96.0	91.7	87.9	90.3	87.5	83.9	91.2



-Conclusion: EfficientPose is a scalable ConvNet architecture leveraging a computationally efficient multi-scale feature extractor, novel mobile detection blocks, skeleton estimation, and bilinear upscaling.

In order to have model variants that are able to flexibly find a sensible trade-off between accuracy and efficiency, we have **exploited model scalability in three dimensions: input resolution, network width, and network depth**.

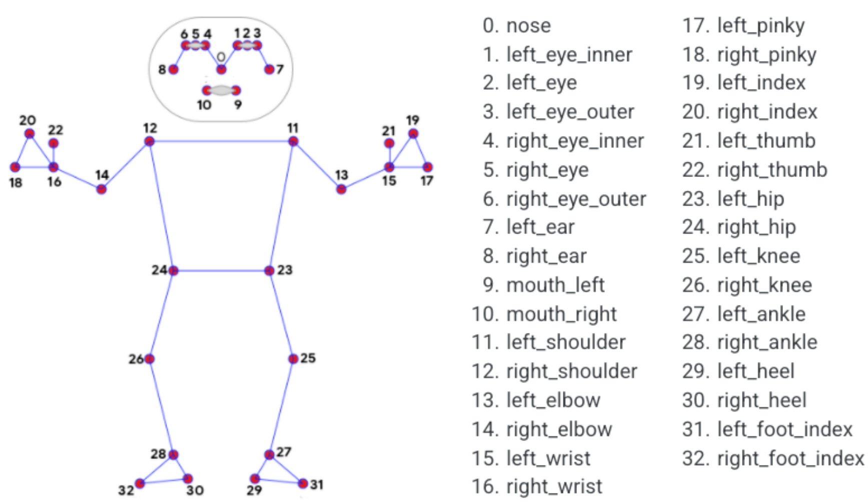
-Run sample code:

```
(base) C:\Users\nguye>conda activate efficientPose

(efficientPose) C:\Users\nguye>cd EfficientPose

(efficientPose) C:\Users\nguye\EfficientPose>python track.py --model=tflite
```

About BlazePose

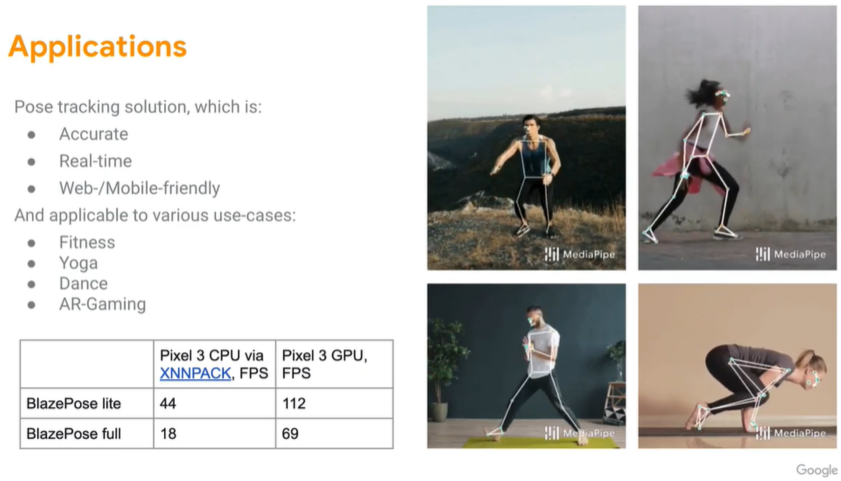
https://arxiv.org/pdf/2006.10204.pdf

-Pose tracking model by Google explitly designed for fitness, yoga, sport, etc.



https://google.github.io/mediapipe/solutions/pose.html

-Application:



https://github.com/geaxgx/openvino_blazepose

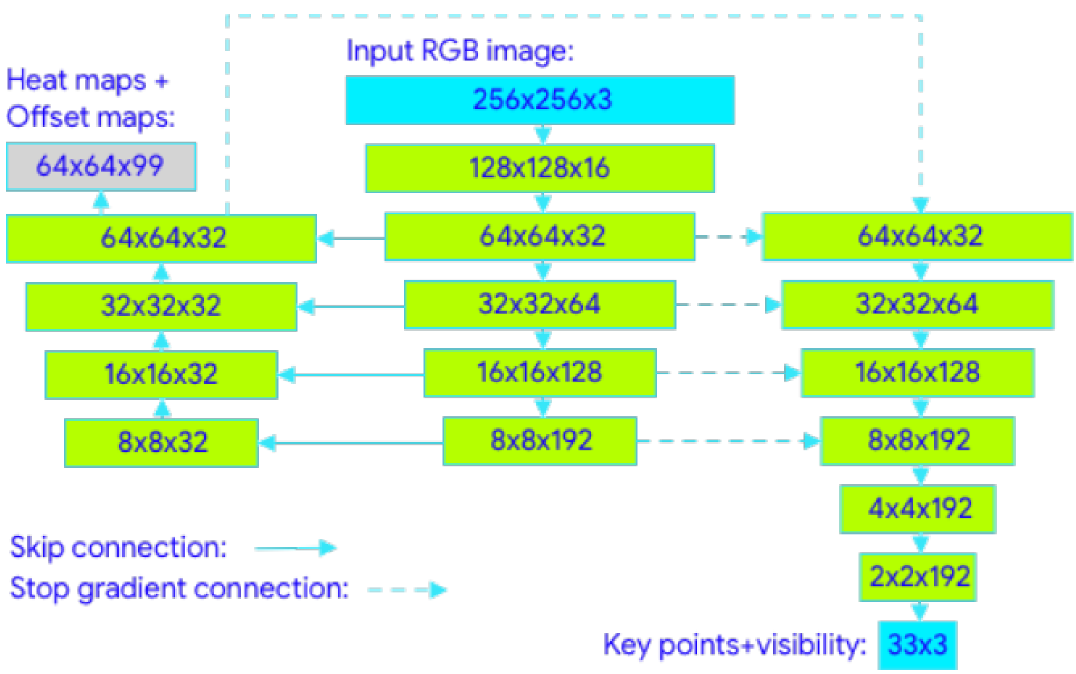


Figure 4. Network architecture. See text for details.

-*BlazePose* consists of two machine learning models: a *Detector* and an *Estimator*

- The *Detector* cuts out the human region from the input image

- The *Estimator* takes a 256x256 resolution image of the detected person as input and outputs the keypoints.

-*BlazePose* outputs the 33 keypoints according the following ordering convention. This is more points than the commonly used 17 keypoints of the COCO dataset.

-The *Detector* is an Single-Shot Detector(SSD) based architecture. Given an **input image** (1,224,224,3), it outputs a **bounding box** (1,2254,12) and a **confidence score** (1,2254,1).

- The 12 elements of the bounding box are of the form (x,y,w,h,kp1x,kp1y,…,kp4x,kp4y), where kp1x to kp4y are additional keypoints. Each one of the 2254 elements has its own anchor, anchor scale and offset need to be applied.

-Comparison:

Method	Yoga mAP	Yoga PCK@0.2
BlazePose GHUM Heavy	68.1	**96.4**
BlazePose GHUM Full	62.6	**95.5**
BlazePose GHUM Lite	45.0	**90.2**
AlphaPose ResNet50	63.4	**96.0**
Apple Vision	32.8	**82.7**

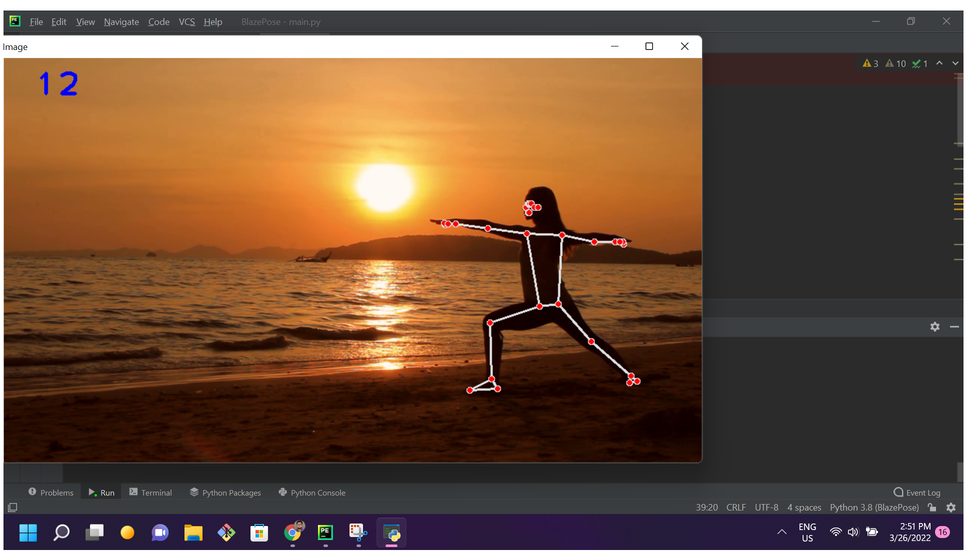
	MacBook Pro 15" 2019. Intel core i9. AMD Radeon Pro Vega 20 Graphics. (FPS)	**iPhone 11** (FPS)	**Pixel 5** (FPS)	**Desktop** Intel i9-10900K. Nvidia GTX 1070 GPU. (FPS)
MediaPipe Runtime With WASM & GPU Accel.	75 | 67 | 34	9 | 6 | N/A	25 | 21 | 8	150 | 130 | 97
TFJS Runtime With WebGL backend.	52 | 40 | 24	43 | 32 | 22	14 | 10 | 4	42 | 35 | 29

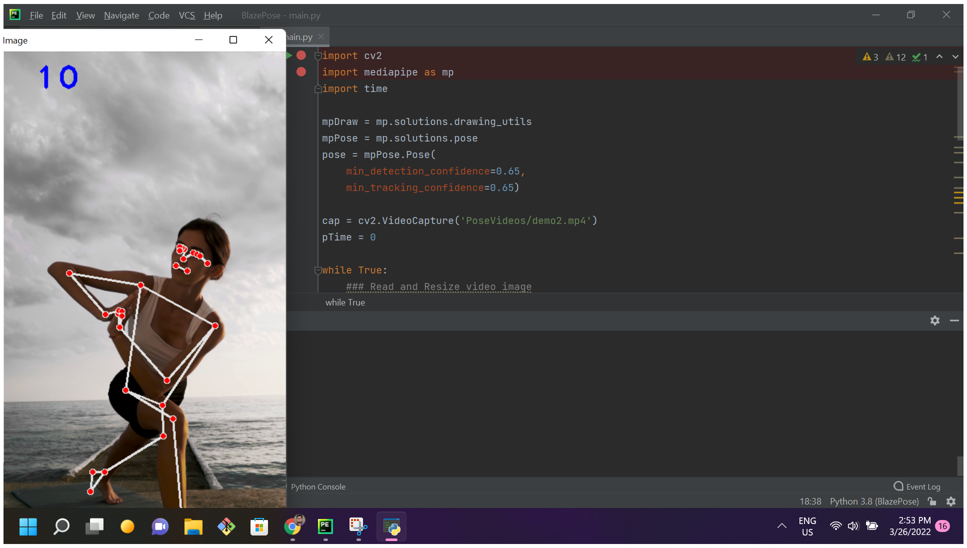
(Source: TensorFlow)

-Code:

```
import cv2
import mediapipe as mp
import time

mpDraw = mp.solutions.drawing_utils
mpPose = mp.solutions.pose
pose = mpPose.Pose(
    min_detection_confidence=0.65,
    min_tracking_confidence=0.65)

cap = cv2.VideoCapture('PoseVideos/demo1.mp4')
pTime = 0
```

```python
while True:
    ### Read and Resize video image
    success, img = cap.read()
    img = cv2.resize(img, (960, 540))
    ##img = cv2.resize(img, (405, 768)) for demo2

    ### Convert to RGB
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = pose.process(imgRGB)
    ## => Framerate decrease

    ### Draw skeleton:
    print(results.pose_landmarks)
    if results.pose_landmarks:
        mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS)

    ### Time capture
    cTime = time.time()
    fps = 1/(cTime-pTime)
    pTime = cTime

    cv2.putText(img, str(int(fps)), (70, 50), cv2.FONT_HERSHEY_PLAIN,3, (255,0,0), 3)
    cv2.imshow("Image", img)
```




File Edit View Navigate Code VCS Help BlazePose - main.py
Image
10
import cv2
import mediapipe as mp
import time
mpDraw = mp.solutions.drawing_utils
mpPose = mp.solutions.pose
pose = mpPose.Pose(
min_detection_confidence=0.65,
min_tracking_confidence=0.65)
cap = cv2.VideoCapture('PoseVideos/demo2.mp4')
pTime = 0
while True:
Read and Resize video image
while True
Python Console
Event Log
18:38 Python 3.8 (BlazePose)
ENG
US
2:53 PM
3/26/2022