

Pose Estimation Using OpenPose

```
In [4]: import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
from zipfile import ZipFile
from urllib.request import urlretrieve

from IPython.display import YouTubeVideo, display, Image

%matplotlib inline
```

```
In [5]: def download_and_unzip(url, save_path):
    print(f"Downloading and extracting assets....", end="")
    # Downloading zip file using urllib package.
    urlretrieve(url, save_path)

    try:
        # Extracting zip file using the zipfile package.
        with ZipFile(save_path) as z:
            # Extract ZIP file contents in the same directory.
            z.extractall(os.path.split(save_path)[0])

        print("Done")
    except Exception as e:
        print("\nInvalid file.", e)
```

```
In [7]: URL = r"https://www.dropbox.com/s/089r2yg6aa08581/opencv_bootcamp_assets_NB14.zip?dl=1"

asset_zip_path = os.path.join(os.getcwd(), "opencv_bootcamp_assets_NB14.zip")

# Download if asset ZIP does not exists.
if not os.path.exists(asset_zip_path):
    download_and_unzip(URL, asset_zip_path)
```

Downloading and extracting assets....Done

Highlights Train elsewhere, perform inference in an OpenCV application

Supports Caffe, Tensorflow, Torch and Darknet.

Supported layers : AbsVal, AveragePooling, BatchNormalization, Concatenation, Convolution (including dilated convolution), Crop, Deconvolution, DetectionOutput (SSD-specific layer), Dropout, Eltwise (+, *, max), Flatten, FullyConnected, LRN, LSTM, MaxPooling, MaxUnpooling, MVN, NormalizeBBox (SSD-specific layer), Padding, Permute, Power, PReLU (including ChannelPReLU with channel-specific slopes), PriorBox (SSD-specific layer), ReLU, RNN, Scale, Shift, Sigmoid, Slice, Softmax, Split, TanH

Use Caffe model trained on the Multi-Person Image Dataset (MPI) to demonstrate human pose estimation for a single person.

```
In [8]: video = YouTubeVideo("RyCsSc_2ZEI", width=1024, height=640)
display(video)
```

Load a Caffe Model A typical Caffe Model has two files

Architecture : Defined in a .prototxt file Weights : Defined in .caffemodel file

```
In [9]: protoFile = "pose_deploy_linevec_faster_4_stages.prototxt"
weightsFile = os.path.join("model", "pose_iter_160000.caffemodel")
```

```
In [10]: nPoints = 15
POSE_PAIRS = [
    [0, 1],
    [1, 2],
    [2, 3],
    [3, 4],
    [4, 5],
    [5, 6],
    [6, 7],
    [7, 14],
    [14, 8],
    [8, 9],
    [9, 10],
    [10, 11],
    [11, 12],
    [12, 13],
]
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)
```

```
In [11]: im = cv2.imread("Tiger_Woods_crop.png")
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)

inWidth = im.shape[1]
inHeight = im.shape[0]
```

In [12]: `Image(filename="Tiger_Woods.png")`

Out[12]:



Convert image to blob

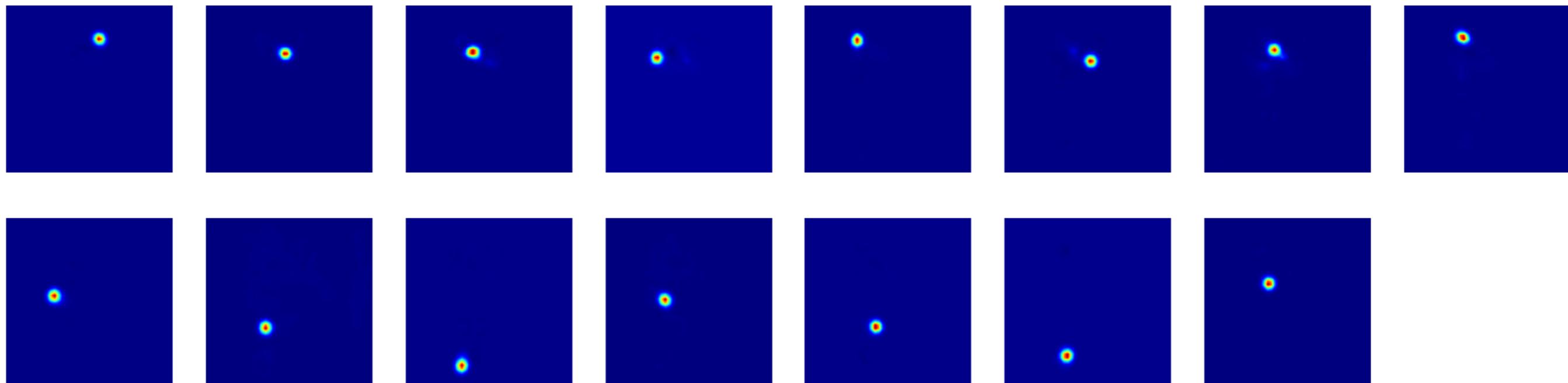
```
In [13]: net.setInputSize = (368, 368)
inpBlob = cv2.dnn.blobFromImage(im, 1.0 / 255, netInputSize, (0, 0, 0), swapRB=True, crop=False)
net.setInput(inpBlob)
```

Run Inference (forward pass)

```
In [14]: # Forward Pass
output = net.forward()

# Display probability maps
plt.figure(figsize=(20, 5))
for i in range(nPoints):
    probMap = output[0, i, :, :]
    displayMap = cv2.resize(probMap, (inWidth, inHeight), cv2.INTER_LINEAR)

    plt.subplot(2, 8, i + 1)
    plt.axis("off")
    plt.imshow(displayMap, cmap="jet")
```



Extract points

In [15]:

```
# X and Y Scale
scaleX = inWidth / output.shape[3]
scaleY = inHeight / output.shape[2]

# Empty list to store the detected keypoints
points = []

# Threshold
threshold = 0.1

for i in range(nPoints):
    # Obtain probability map
    probMap = output[0, i, :, :]

    # Find global maxima of the probMap.
    minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

    # Scale the point to fit on the original image
    x = scaleX * point[0]
    y = scaleY * point[1]

    if prob > threshold:
        # Add the point to the list if the probability is greater than the threshold
        points.append((int(x), int(y)))
    else:
        points.append(None)
```

Display Points & Skeleton

In [16]:

```
imPoints = im.copy()
imSkeleton = im.copy()

# Draw points
for i, p in enumerate(points):
    cv2.circle(imPoints, p, 8, (255, 255, 0), thickness=-1, lineType=cv2.FILLED)
    cv2.putText(imPoints, "{}".format(i), p, cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, lineType=cv2.LINE_AA)

# Draw skeleton
for pair in POSE_PAIRS:
    partA = pair[0]
    partB = pair[1]

    if points[partA] and points[partB]:
        cv2.line(imSkeleton, points[partA], points[partB], (255, 255, 0), 2)
        cv2.circle(imSkeleton, points[partA], 8, (255, 0, 0), thickness=-1, lineType=cv2.FILLED)
```

```
In [17]: plt.figure(figsize=(50, 50))

plt.subplot(121)
plt.axis("off")
plt.imshow(imPoints)

plt.subplot(122)
plt.axis("off")
plt.imshow(imSkeleton)
```

Out[17]: <matplotlib.image.AxesImage at 0x17a2df75110>



In [18]: Image(filename="Milton_Golf_Swing.png")

Out[18]:



In []: