Backbone refers to the feature extraction network used in a neural network architecture. For understanding this better let us take the example of Action transformer which is built on top of a pose estimation model.

Action Transformer Overview (https://arxiv.org/abs/2107.00606)

- The Transformer architecture has been one of the most important deep learning advances of the last years in natural language processing
- Human Action Recognition have proposed the integration of attention mechanisms with convolutional and recurrent blocks to improve the accuracy of models.
- Action Transformer applies a pure Transformer encoder derived architecture to action recognition obtaining an accurate and low-latency model for real-time applications.

Switching Backbone

- Before we make changes to the backbone which extracts features, we need understand the input shape for the transformer.

```
=================================================================
input_3 (InputLayer)        [(None, 30, 52)]          0

_________________________________________________________________
dense_78 (Dense)             (None, 30, 64)            3392

_________________________________________________________________
patch_class_embedding_2 (Pat (None, 31, 64)            2048

_________________________________________________________________
transformer_encoder_2 (Trans (None, 31, 64)            199936

_________________________________________________________________
lambda_2 (Lambda)            (None, 64)                0

_________________________________________________________________
dense_79 (Dense)             (None, 256)               16640

_________________________________________________________________
dense_80 (Dense)             (None, 20)                5140
=================================================================
Total params: 227,156
Trainable params: 227,156
Non-trainable params: 0
```

- We need a pose estimation model that outputs the same dimension n x 30 x 52 and for this example we use open pose (https://github.com/CMU-Perceptual-Computing-Lab/openpose)

- In the next step we change the dimensions outputted by the pose estimation model so it is similar to the input used for the Action Transformer.

```
BODY_PARTS = { "Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4,
               "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9,
               "RAnkle": 10, "LHip": 11, "LKnee": 12, "LAnkle": 13, "REye": 14,
               "LEye": 15, "REar": 16, "LEar": 17, "Background": 18 }

POSE_PAIRS = [ ["Neck", "RShoulder"], ["Neck", "LShoulder"], ["RShoulder", "RElbow"],
               ["RElbow", "RWrist"], ["LShoulder", "LElbow"], ["LElbow", "LWrist"],
               ["Neck", "RHip"], ["RHip", "RKnee"], ["RKnee", "RAnkle"], ["Neck", "LHip"],
               ["LHip", "LKnee"], ["LKnee", "LAnkle"], ["Neck", "Nose"], ["Nose", "REye"],
               ["REye", "REar"], ["Nose", "LEye"], ["LEye", "LEar"] ]
```

n x 30 x 19 x 2 is the dimension of the output data from the pose estimation model for the first 30 frames. We need to alter the dimensions so that it can be fed into the transformer.

(number_of_samples, time_window, number_of_keypoints, x_y)

Number_of_samples = n

Time_window = 30

Number_of_keypoints = 19

X_y = 2D keypoint coordinates.

```
In [10]: import numpy as np
         print(np.shape(list_30f))
         print(list_30f)
```

```
(4, 30, 19, 2)
[[[[806, 140], [862, 187], [806, 187], [779, 281], [0, 0], [890, 203], [834, 281], [806, 219], [806, 344], [0, 0], [723, 64
1], [862, 344], [0, 0], [0, 0], [0, 0], [834, 125], [0, 0], [862, 125], [1224, 15]], [[806, 125], [862, 187], [806, 187], [77
9, 266], [0, 0], [890, 203], [806, 281], [806, 219], [806, 360], [0, 0], [723, 641], [862, 344], [0, 0], [0, 0], [0, 0], [83
4, 125], [0, 0], [862, 125], [1224, 15]], [[834, 125], [862, 187], [834, 187], [0, 0], [0, 0], [890, 203], [806, 281], [806,
219], [834, 360], [0, 0], [723, 641], [862, 344], [0, 0], [0, 0], [0, 0], [834, 125], [0, 0], [862, 125], [1224, 15]], [[834,
125], [862, 187], [834, 187], [0, 0], [0, 0], [890, 203], [806, 281], [806, 219], [806, 360], [0, 0], [723, 641], [862, 344],
[0, 0], [0, 0], [0, 0], [834, 125], [0, 0], [862, 125], [1224, 15]], [[834, 125], [862, 187], [890, 187], [0, 0], [0, 0], [89
0, 203], [806, 281], [806, 219], [834, 360], [0, 0], [723, 641], [862, 344], [0, 0], [0, 0], [0, 0], [834, 125], [0, 0], [86
2, 125], [1224, 15]], [[834, 125], [862, 203], [890, 187], [0, 0], [0, 0], [890, 203], [806, 281], [806, 219], [834, 344],
[0, 0], [723, 641], [862, 344], [0, 0], [723, 641], [0, 0], [834, 125], [0, 0], [862, 125], [166, 0]], [[0, 0], [862, 203],
[890, 187], [806, 281], [0, 0], [890, 203], [806, 281], [806, 219], [834, 360], [0, 0], [723, 641], [862, 344], [0, 0], [0,
0], [0, 0], [834, 125], [0, 0], [862, 125], [166, 0]], [[0, 0], [862, 203], [890, 187], [806, 281], [0, 0], [890, 203], [806,
281], [806, 219], [834, 360], [0, 0], [723, 641], [862, 344], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [166, 0]], [[0,
0], [862, 203], [890, 187], [779, 281], [0, 0], [890, 203], [806, 281], [806, 219], [834, 344], [0, 0], [723, 641], [862, 34
4], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [166, 0]], [[0, 0], [862, 187], [890, 187], [0, 0], [0, 0], [890, 203],
[806, 281], [806, 219], [862, 344], [0, 0], [723, 641], [862, 344], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [862, 125], [166,
0]], [[0, 0], [862, 187], [890, 187], [0, 0], [0, 0], [890, 203], [806, 281], [806, 219], [862, 344], [0, 0], [723, 641], [86
2, 344], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [862, 125], [166, 0]], [[0, 0], [862, 187], [890, 187], [0, 0], [0, 0], [89
```

1. Re ordering indices: After reordering we can see that we adjusted the pose pairs after processing the key points. n x 30 17 x 2

In [8]:
```python
import itertools

list_a = []
list_b = []

for num in range (len(list_30f)):
    for frame in range(0,30):
        reorder_indices = [0, 15, 14, 17, 16, 5, 2, 6, 3, 7, 4, 11, 8, 12, 9, 13, 10]
        new_points = [list_30f[num][frame][i] for i in reorder_indices]
        list_a.append(new_points)
    print(np.shape(list_a))
    list_b.append(list_a)
    list_a=[]
print(np.shape(list_b))
print(list_b)
```

```
(2, 30, 17, 2)
[[[[806, 140], [834, 125], [0, 0], [862, 125], [0, 0], [890, 203], [806, 187], [834, 281], [779, 281], [806, 219], [0, 0], [8
62, 344], [806, 344], [0, 0], [0, 0], [0, 0], [723, 641]], [[806, 125], [834, 125], [0, 0], [862, 125], [0, 0], [890, 203],
[806, 187], [806, 281], [779, 266], [806, 219], [0, 0], [862, 344], [806, 360], [0, 0], [0, 0], [0, 0], [723, 641]], [[834, 1
25], [834, 125], [0, 0], [862, 125], [0, 0], [890, 203], [834, 187], [806, 281], [0, 0], [806, 219], [0, 0], [862, 344], [83
4, 360], [0, 0], [0, 0], [0, 0], [723, 641]], [[834, 125], [834, 125], [0, 0], [862, 125], [0, 0], [890, 203], [834, 187], [8
06, 281], [0, 0], [806, 219], [0, 0], [862, 344], [806, 360], [0, 0], [0, 0], [0, 0], [723, 641]], [[834, 125], [834, 125],
```

The following three steps have methods as part of the source code for action transformer which can be used to adjust our data.

2. Reducing key points: Reduces the key points from 17 to 13 and the output dimension is n x 30 x 13 x 2.

```python
def reduce_keypoints(arr):
        if np.shape(arr)[2] <= 15:
            print('Keypoint number has already been reduced!')
            return
        seq_list = []
        to_prune = []
        h= [0,1,2,3,4]
        rf=[15]
        lf=[16]

        for group in [h, rf, lf]:
            if len(group) > 1:
                to_prune.append(group[1:])
        to_prune = [item for sublist in to_prune for item in sublist]

        for seq in arr:
            seq[:,h[0],:] = np.true_divide(seq[:,h,:].sum(1), (seq[:,h,:] != 0).sum(1)+1e-9)
            seq[:,rf[0],:] = np.true_divide(seq[:,rf,:].sum(1), (seq[:,rf,:] != 0).sum(1)+1e-9)
            seq[:,lf[0],:] = np.true_divide(seq[:,lf,:].sum(1), (seq[:,lf,:] != 0).sum(1)+1e-9)
            seq_list.append(seq)
        arr = np.stack(seq_list)
        arr = np.delete(arr, to_prune, 2)
        print("*** reduced keypoints ***",np.shape(arr))
        return arr
```

3. Scale and center: We scale and center our key points and our output dimension remains the same. n x 30 x 13 x 2

```
def scale_and_center(arr):
        arr = [arr]
        for X in [arr]:
            seq_list = []
            for seq in X:
                pose_list = []
                for pose in seq:
                    zero_point = (pose[1, :2] + pose[2,:2]) / 2
                    module_keypoint = (pose[7, :2] + pose[8,:2]) / 2
                    scale_mag = np.linalg.norm(zero_point - module_keypoint)
                    if scale_mag < 1:
                        scale_mag = 1
                    pose[:,:2] = (pose[:,:2] - zero_point) / scale_mag
                    pose_list.append(pose)
                seq = np.stack(pose_list)
                seq_list.append(seq)
            X = np.stack(seq_list)

        arr = np.delete(arr,[], 2)
        arr = np.squeeze(arr)
        print('\n *** scale and center *** \n')
        print(arr.shape)
        return arr
```

4. Add velocities: After adding velocities extra dimensions gets added and our output dimensions are n x 30 x 13 x 4.

```
def add_velocities(seq, T=30, C=3):
    v1 = np.zeros((T+1, seq.shape[1], C-1))
    v2 = np.zeros((T+1, seq.shape[1], C-1))
    v1[1:,...] = seq[:,:,:2]
    v2[:T,...] = seq[:,:,:2]
    vel = (v2-v1)[:-1,...]
    data = np.concatenate((seq[:,:,:2], vel), axis=-1)
    print("*** add vel ***",np.shape(data))
    return data
```

5. Reshaping: Then we reshape as per the model which makes us land with the final dimensions n x 30 x 52 which can then be fed into our model for predictions.

```
addvel = addvel.reshape(1, 30,-1)
```