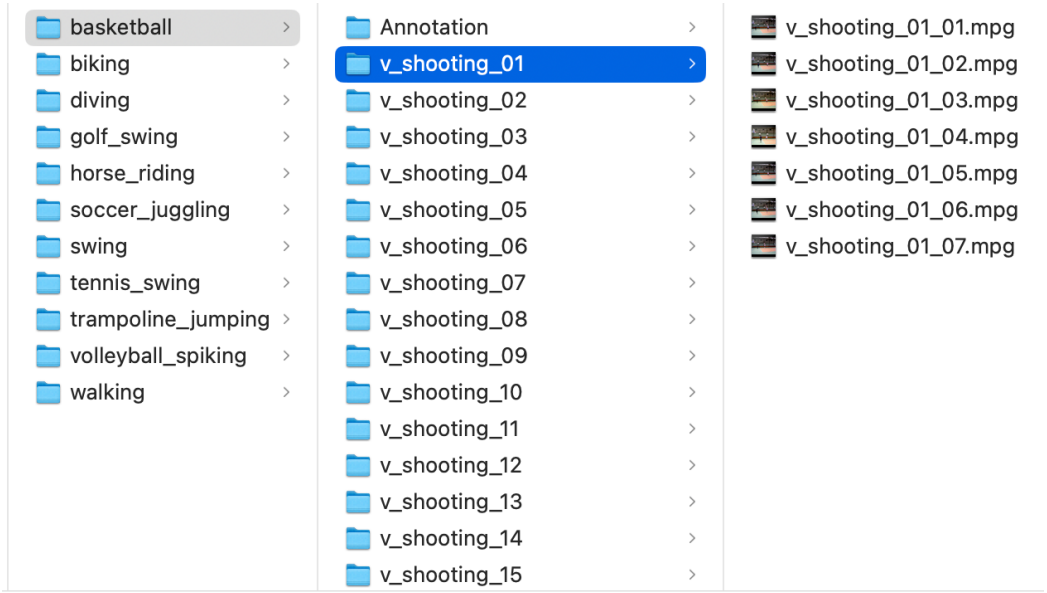
Exercise
Human Action Recognition

In this exercise, you will design your **deep model** for human action recognition from videos. This task is an example of video classification.

Dataset:
You will use the **UCF11 dataset** (https://www.crcv.ucf.edu/data/UCF_YouTube_Action.php).
It contains **11 action categories**: *basketball shooting, biking/cycling, diving, golf swinging, horse back riding, soccer juggling, swinging, tennis swinging, trampoline jumping, volleyball spiking, and walking with a dog.*
This data set is challenging due to large variations in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, illumination conditions, etc.
For each category, **the videos are grouped into 25 groups with more than 4 action clips in it**. The video clips in the same group share some common features, such as the same actor, similar background, similar viewpoint, and so on.

Data Preparation:
Once you download and unzip the file, you will have this directory tree:



You may get the list of files and corresponding labels with the following code

```
def load_groups(input_folder):
    '''
    Loading the list of sub-folders into a python list with their
    corresponding label.
    '''
    groups          = []
    label_folders   = os.listdir(input_folder)
    index           = 0
    for label_folder in sorted(label_folders):
        label_folder_path = os.path.join(input_folder, label_folder)
        if os.path.isdir(label_folder_path):
            group_folders = os.listdir(label_folder_path)
            for group_folder in group_folders:
                if group_folder != 'Annotation':
                    groups.append([os.path.join(label_folder_path,
                                                group_folder), index])
            index += 1

    return groups
```

You may wanna consider files with more than 16 frames.
You can split the dataset into training and test set as it follows:

```
def get_file_list(train_groups):
    train = []
    for video_dir in train_groups:
        video_files = os.listdir(video_dir[0])
        for video_file in video_files:
            video_file_path = os.path.join(video_dir[0], video_file)
            if os.path.isfile(video_file_path):
                ext = os.path.splitext(video_file_path)[1]
                if (ext == '.mpg'):
                    # make sure we have enough frames and the file isn't corrupt
                    video_reader = cv2.VideoCapture(video_file_path)
                    if video_reader and
```

```
                                video_reader.get(cv2.CAP_PROP_FRAME_COUNT) >= 16:
                            train.append([video_file_path, video_dir[1]])
    return train

def split_data(groups):
    '''
    Splitting the data at random for train and test set.
    '''
    group_count = len(groups)
    indices = np.arange(group_count)

    np.random.seed(0) # Make it deterministic.
    np.random.shuffle(indices)

    # 60% training, 20% val and 20% test.
    train_count = int(0.6 * group_count)
    val_ind = int(0.8 * group_count)

    train_groups = [groups[i] for i in indices[:train_count]]
    val_groups  = [groups[i] for i in indices[train_count:val_ind]]
    test_groups  = [groups[i] for i in indices[val_ind:]]

    train = get_file_list(train_groups)
    val = get_file_list(val_groups)
    test = get_file_list(test_groups)

    return train, val, test
```

The code to read and split the data is:

```
groups = load_groups('UCF11_updated_mpg')
train, val, test = split_data(groups)
```

Train, validation and test sets will be lists similar to the following:

[['UCF11/soccer_juggling/v_juggle_18/v_juggle_18_06.mpg', 5], ['UCF11/soccer_juggling/v_juggle_18/v_juggle_18_07.mpg', 5], …]

You should also prepare a data generator to read the images in each video, rescale to a proper size (maybe 64 x 64) and, depending on the model you wanna use, subsample (perhaps 16 frames?). Don't' forget to cast to float32 and use some data pre-processing.

Design the model:

A video consists of an ordered sequence of frames. Each frame contains spatial information, and the sequence of those frames contains temporal information. To model these aspects, it is generally needed to implement hybrid architecture that consists of convolutions (**for spatial processing**) as well as some technique to deal with the **temporal information**. An approach may be the use of recurrent layers (check the keras documentation to explore different layers. Sometimes more recurrent layers are also concatenated).

If you wish, you are also free to explore other techniques, such as Conv3D or feature map reshaping; in these cases, be careful with the length of the frame sequence. (*There is no right or wrong model*).

You may wanna also experiment with some pre-trained convolutional backbone. For instance, first few blocks of a pre-trained residual net or maybe InceptionV3?

If you are unsure, use the first few layers of a pre-trained model (maybe resnet-50) and then a ConvLSTM cell.

For instance, the following code creates a backbone with the first two residual blocks of a ResNet50 (a block ends after the activation of the sum within the residual block, see slides)

```
def get_backbone(i_s = (64, 64, 3)):
    baseModel = ResNet50(weights='imagenet', include_top=False)
    res =  Model(inputs=baseModel.input,
                outputs=baseModel.get_layer('conv2_block2_out').output)

    input = Input(i_s)
    input_p = tf.keras.applications.resnet50.preprocess_input(input)
    output = res(input_p)
    model = Model(inputs=input, outputs=output)
    model.summary()
    return model
```

Performance:
After the training is complete, you will evaluate the model on your test set.
You should prepare a table comparing the accuracy values in test for your developed models. This can be done by including in the **compile** method the attribute **metrics=[tf.keras.metrics.CategoricalAccuracy()].** The table should also report the total number of parameters in your models, the final training loss, the final validation loss, and the final test loss.