Problem - 1:
Suppose you have a dataset containing video clips. Each clip shows a single action been performed. The dataset contains four actions - running, jumping, standing still, kicking a ball.

Your task is to suggest a model, which will help us to classify the clips based on the action performed in the clip.

Example if we input to the model, a clip performing running action, the model should output the class as 'running'.
You are expected to define your approach in terms of input to the model, model architecture and what kind of output the model generates. You are not required to program anything. You have the liberty to refer to various literatures and researches on the same problem and your submission could even be a summary of your exploration.

The Model I would recommend is an InceptionV3 OR VGG-16 (Whichever works better for our dataset) as my base model followed by Keras sequential API to fine tune according to the 4 class labels specified.

Overview of Steps to Classify video clips based on the action performed in the Clip

1. Choosing a data set to train.
2. Segment or Extracting Frames from the Dataset of Videos.
3. Extract Feature from the frames of data
4. Pre-process the Features frames.
5. Create a model architecture over the base model and Fine tuning of model's head
6. Train the Model and save weights
7. Predict to classify our Test examples
8. Evaluate to improve/fine tune further

Further Exploration Areas

1. Choosing a data set to train.

The action recognition problem requires huge computational costs and lots of data. Fortunately, several very good datasets are available/ Together with the previously available benchmarks (ActivityNet, UCF101, HMDB), they build a great foundation for significant improvements in the performance of the action recognition system.
Thus many available video data sets are used to train the models and fine tune according to the required label set which in this case are : **running, jumping, standing still, kicking a ball.**

Link to paper with 15+ datasets to Choose from
Link to HowTo100M

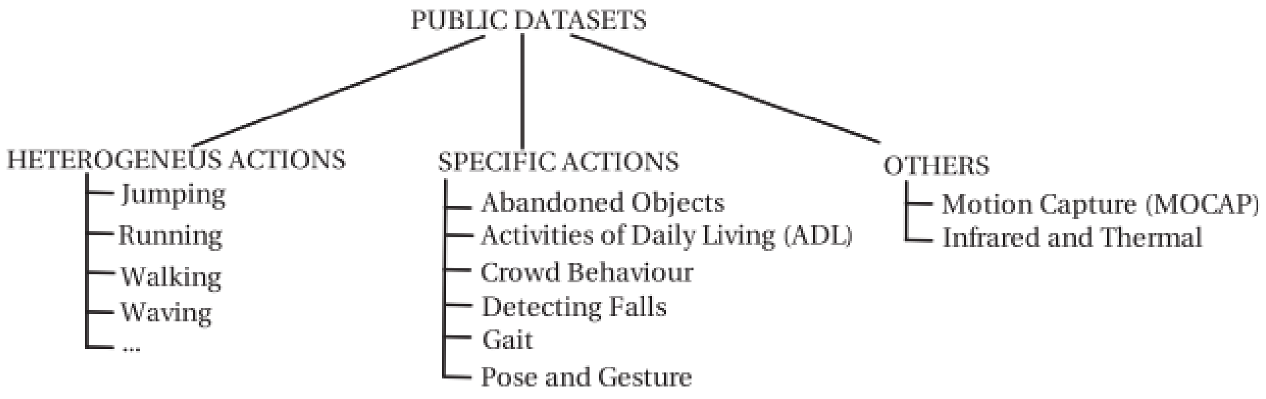

Fig. 1. A possible taxonomy of the datasets according to the type of actions.

One approach to get started is to merge the Available Video data sets with the self-generated datasets produced in the environment where the model will be eventually applied and then Train/Test split the data for Training, validation and test sets so that the data is distributed normally and with unbaised divisions. This can result in large amount of data as well as specification resulting in more accurate

Also, from the online datasets, we can filter out frames with label anme only consisting or similar to our 4 required labels to build a less generic and more accurate model.

2. Segment or Extract Frames from the Dataset of Videos.

Using the **OpenCV** library we can capture frame-by-frame data from the videos data set for training as well as validation set

Further exploration can include :
There are several possible ways of Extracting the frames that can assist in increasing accuracy:

Approach 1: Single Stream Network

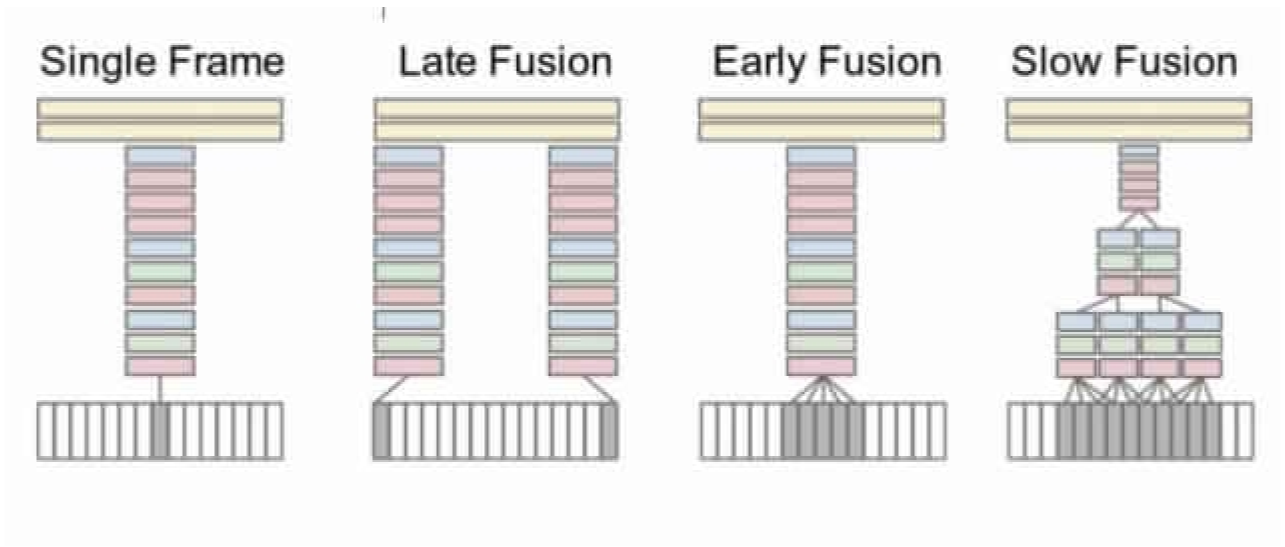

Figure 1: Image Sourse

Approach 2: Two Stream Networks

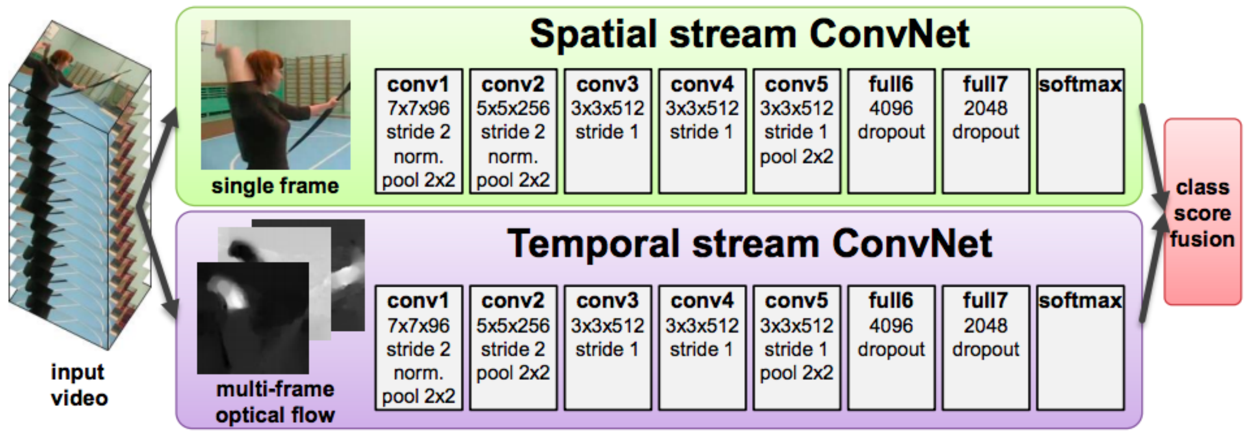

Figure 2: Image Source

Following papers which are, in a way, evolution from the two papers (single stream and two stream) which are summarized as below:
LRCN, C3D, Conv3D & Attention, TwoStreamFusion,TSN, ActionVlad, HiddenTwoStream, I3D, T3D

More on Two-StreamFusion

3. Extract Features from the frames of videos

Extracting features/weights from an existing trained model is a process of transfer learning that helps in identifying the high level features in the earlier stages of the model. Many such applications/ models are provided by Keras namely [*Xception*, ***VGG16***, *VGG19*, *ResNet*, *ResNetV2*, *ResNeXt*, ***InceptionV3***, *InceptionResNetV2*, *MobileNet*, *MobileNetV2*, *DenseNet*, *NASNet]*

As per my exploratory research I would recommend an InceptionV3 OR VGG16 (choosing which works better for our dataset) as the **base model** to extract features from our segmented train/validation/test frames.

One important note while using the base_model is to **set the top of the model as false so that we can fine tune the same as per our needs.** `include_top = False` will remove the last layer of this model so that we can tune it as per our need. This process is 'Fine Tuning' [*Refer Figure 4 & 5*]

Following is an image of VGG-16 and VGG-19 and we can see, a*t the end we have final 7×7×512 into Fullyconnected layer (FC) with 4096 units, and in a softmax output one of a 1000 classes.*

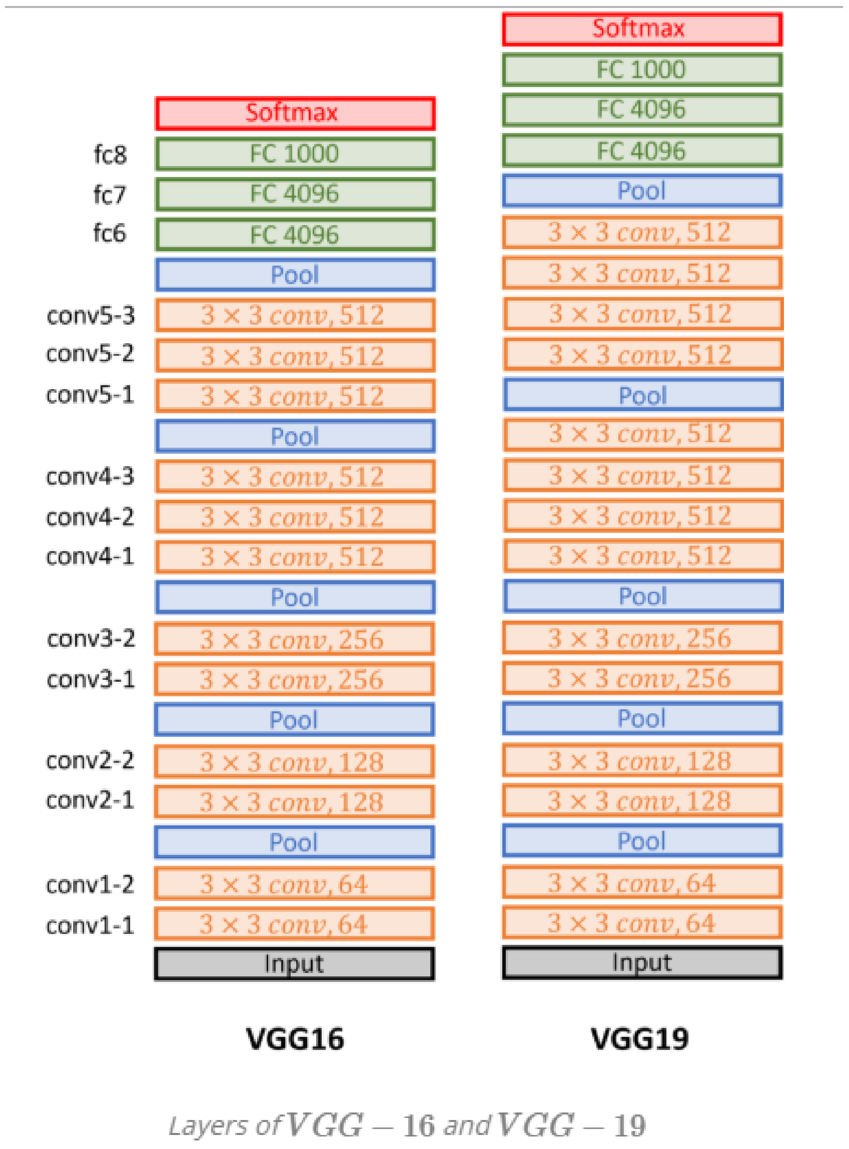

Figure 3: Image Source

4. Pre-process the Features frames.

These are pre-processing steps.
Use a fully connected network now to fine-tune the model will take input in single dimension. So, we will reshape the images into a single dimension:

```
reshape(No_Of_samples, 7*7*512)
```

Output: (No_Of_samples, 25088)

It is always advisable to normalize the pixel values, i.e., keep the pixel values between 0 and 1. This helps the **model to converge faster.**

We can also perform data **augmentation** using ImageDataGenerator object provided by Keras, to increase the data-sets size as well to add some distortion.

5. Create a model architecture over the base model and Fine tuning of model's head

Creating our model above the base model's output using either Keras Sequential API or tensorflow.

This recommended model architecture includes :
- Average Pooling
- Flattening of the layer
- Multiple Fully connected Dense layers with relu activation function with Dropout to reduce Over-fitting.
- Final Dense layer with neurons equal to the length of classes i,e **4** and activation as **Softmax (multi-label classification)**

```
model = Model(inputs=baseModel.input, outputs=headModel)
```

We'll now **freeze the baseModel layers** so that it will not be trained via backpropagation. Further we can fine tune few top layers from the base model to increase accuracy(Fine-tuning)

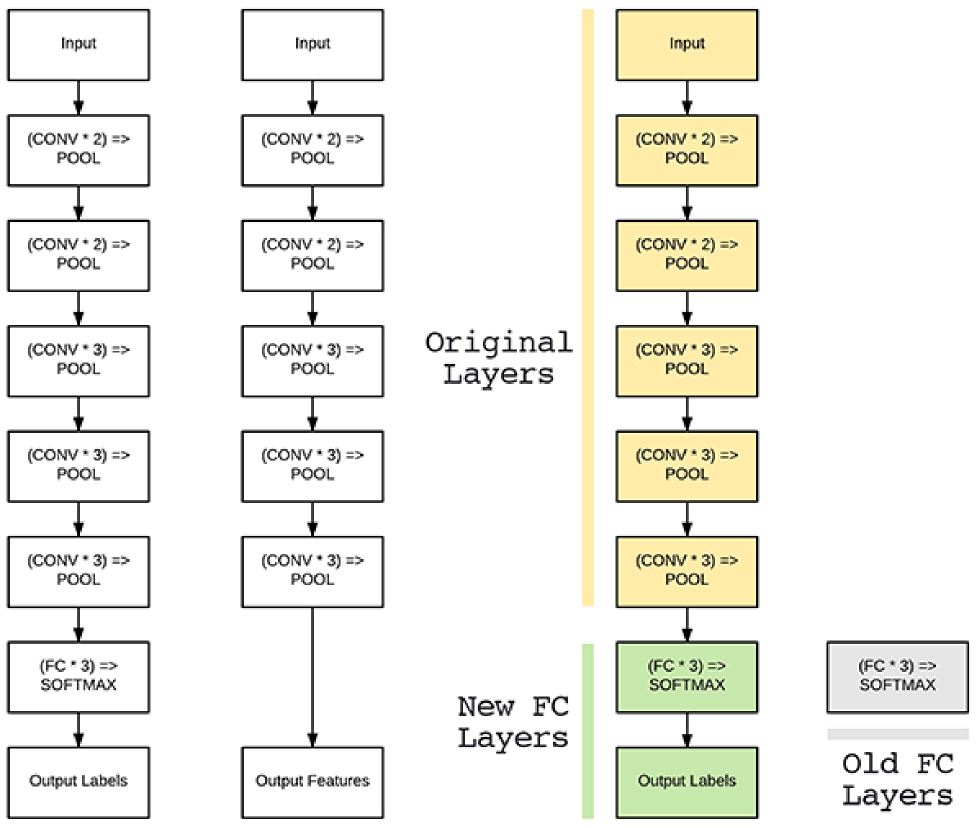

Figure 4: Left: The original VGG16 network architecture. Middle: Removing the FC layers from VGG16 and treating the final POOL layer as a feature extractor. Right: Removing the original FC Layers and replacing them with a brand new FC head. These FC layers can then be fine-tuned to a specific dataset (the old FC Layers are no longer used).

6. Train the Model and save weights

Basic steps in training the model includes **compiling** and **fitting** the model to our X and Y training features set.
Optimization : Using SGD with Momentum or Adam optimizer whichever suits best.
Metrics : accuracy
loss = **categorical_crossentropy** (since it is a multi-class classification problem)

To **save weights** use Keras callbacks ModelCheckpoint to save the best weigths that are learned from training the model we built for use while evaluating the model using the Test data samples.

7. Predict to classify our Test examples

Output of the model can be in several ways, I propose two from my exploration :

A) A DataFrame consisting of the predicted labels with respect to the file names of test set. Comparing them with actual labels can be our evaluation step.
   - **i.** We will take each video from the test set, extract frames for this video.
   - **ii.** Extract features for these frames using the pre-trained model
   - **iii.** Predict tags, and then take the mode to assign a tag for that particular video and append it in the dataframe.

B) Again using the OpenCV Library we draw the prediction on the output frame i.e on the Video clip (**VideoWriter**).

8. Evaluate to improve/fine tune further

Evaluating the accuracy of the model and further tune the model as per the required accuracy depending on the criticality of the application and retrain the model by **unfreezing the earlier layers** in the Pre-Trained model used.

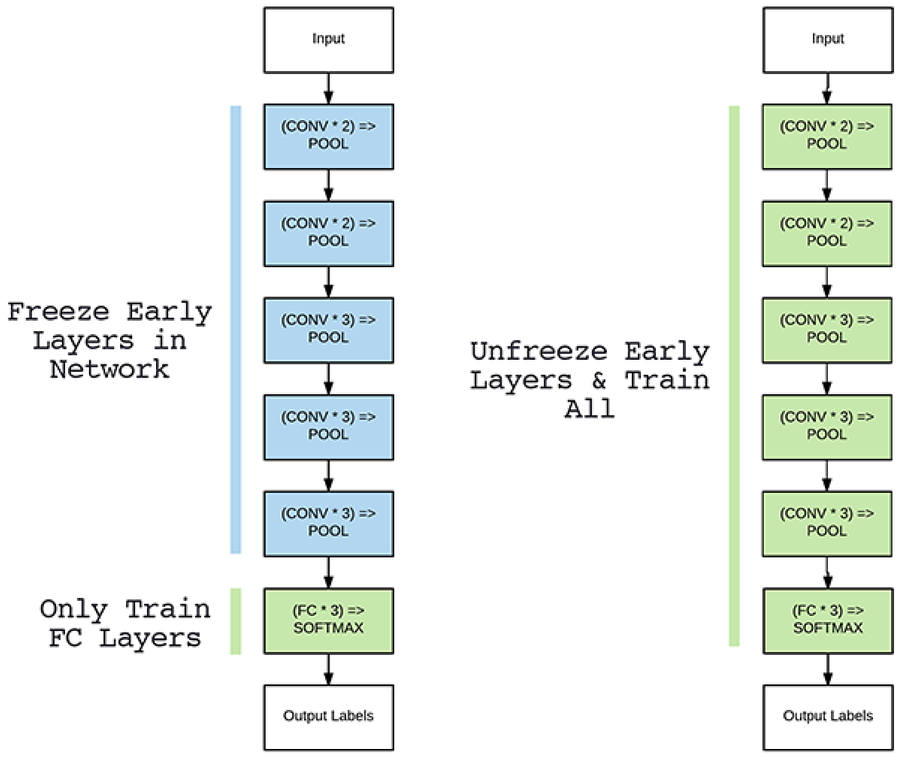

Figure 5: Left: When we start the fine-tuning process, we freeze all CONV layers in the network and only allow the gradient to backpropagate through the FC layers. Doing this allows our network to "warm up". Right: After the FC layers have had a chance to warm up, we may choose to unfreeze all or some of the layers earlier in the network and allow each of them to be fine-tuned as well.

A simple hack to avoid 'Prediction Flickering' is to utilize a **rolling prediction average.** Which is :

- Maintain a list of the last K predictions
- Compute the average of the last K predictions and choose the label with the largest corresponding probability

Further Exploration Areas

1. Use of LSTM + CNN model to also capture the Motion of the frames resulting in even better accuracy.
I decided to not propose the LSTM Model because of its complexity and computation cost which, I believe, is not needed for an application with a small classification set and can be equivalently achieved with a CNN model itself.
Long-term Recurrent Convolutional Networks forVisual Recognition and Description

2. Live video streaming over network with OpenCV and ImageZMQ

3. Real-time Action Recognition using a 3D CNN

4. We can convert any image classification model into video classification model just by adding **timedistributed** layer Refer

5. **Multiresolution CNNs**

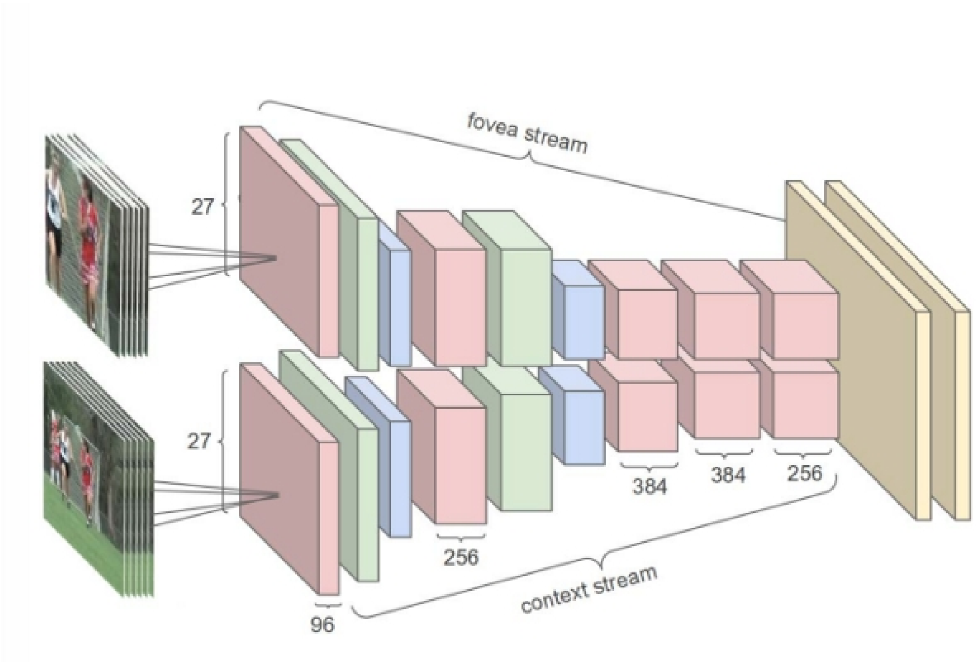

Figure 6: Multiresolution CNNs

6. GitHub Repo For Collection of Computer Vision related work

7. Simple but effective modified LCS approach

7. My previous Work on ResNet Neural Network through Online Courses is as follows.
ResNet for Image Classification of Number through Hand recognition Link