SlowFast model architectures are based on [1] with pretrained weights using the 8x8 setting on the Kinetics dataset.

[1] Christoph Feichtenhofer et al, “SlowFast Networks for Video Recognition”
https://arxiv.org/pdf/1812.03982.pdf (https://arxiv.org/pdf/1812.03982.pdf)

Load the model:

In [22]:
```
# import torch
import torch.fx
!pip install fvcore
# Choose the `slowfast_r50` model
model = torch.hub.load('facebookresearch/pytorchvideo', 'slowfast_r50', pretraine
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: fvcore in c:\users\zeeshan\appdata\roaming\pytho
n\python39\site-packages (0.1.5.post20220512)
Requirement already satisfied: yacs>=0.1.6 in c:\users\zeeshan\appdata\roaming
\python\python39\site-packages (from fvcore) (0.1.8)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\site-packag
es (from fvcore) (4.64.0)
Requirement already satisfied: Pillow in c:\programdata\anaconda3\lib\site-pack
ages (from fvcore) (9.0.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packa
ges (from fvcore) (1.21.5)
Requirement already satisfied: iopath>=0.1.7 in c:\users\zeeshan\appdata\roamin
g\python\python39\site-packages (from fvcore) (0.1.10)
Requirement already satisfied: termcolor>=1.1 in c:\users\zeeshan\appdata\roami
ng\python\python39\site-packages (from fvcore) (1.1.0)
Requirement already satisfied: tabulate in c:\programdata\anaconda3\lib\site-pa
ckages (from fvcore) (0.8.9)
Requirement already satisfied: pyyaml>=5.1 in c:\programdata\anaconda3\lib\site
-packages (from fvcore) (6.0)
Requirement already satisfied: typing-extensions in c:\programdata\anaconda3\li
b\site-packages (from iopath>=0.1.7->fvcore) (4.1.1)
Requirement already satisfied: portalocker in c:\users\zeeshan\appdata\roaming
\python\python39\site-packages (from iopath>=0.1.7->fvcore) (2.5.1)
Requirement already satisfied: pywin32>=226 in c:\programdata\anaconda3\lib\sit
e-packages (from portalocker->iopath>=0.1.7->fvcore) (302)
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-pa
ckages (from tqdm->fvcore) (0.4.4)

Using cache found in C:\Users\zeeshan/.cache\torch\hub\facebookresearch_pytorch
video_main
```

Import remaining functions:

In [23]:
```
from typing import Dict
import json
import urllib
#pip install av


from torchvision.transforms import Compose, Lambda
from torchvision.transforms._transforms_video import (
    CenterCropVideo,
    NormalizeVideo,
)
from pytorchvideo.data.encoded_video import EncodedVideo
from pytorchvideo.transforms import (
    ApplyTransformToKey,
    ShortSideScale,
    UniformTemporalSubsample,
    UniformCropVideo
)
```

Set the model to eval mode and move to desired device.

In [24]:
```
# Set to GPU or CPU
device = "cpu"
model = model.eval()
model = model.to(device)
```

Download the id to label mapping for the Kinetics 400 dataset on which the torch hub models were trained.

This will be used to get the category label names from the predicted class ids.

In [25]:
```
json_url = "https://dl.fbaipublicfiles.com/pyslowfast/dataset/class_names/kinetic
json_filename = "kinetics_classnames.json"
try: urllib.URLopener().retrieve(json_url, json_filename)
except: urllib.request.urlretrieve(json_url, json_filename)
```

In [26]:
```
with open(json_filename, "r") as f:
    kinetics_classnames = json.load(f)

# Create an id to label name mapping
kinetics_id_to_classname = {}
for k, v in kinetics_classnames.items():
    kinetics_id_to_classname[v] = str(k).replace('"', "")
```

Define input transform

Before passing the video into the model we need to apply some input transforms and sample a clip of the correct duration.

In [27]:
```
side_size = 256
mean = [0.45, 0.45, 0.45]
std = [0.225, 0.225, 0.225]
crop_size = 256
num_frames = 32
sampling_rate = 2
frames_per_second = 30
slowfast_alpha = 4
num_clips = 10
num_crops = 3

class PackPathway(torch.nn.Module):
    """
    Transform for converting video frames as a list of tensors.
    """
    def __init__(self):
        super().__init__()

    def forward(self, frames: torch.Tensor):
        fast_pathway = frames
        # Perform temporal sampling from the fast pathway.
        slow_pathway = torch.index_select(
            frames,
            1,
            torch.linspace(
                0, frames.shape[1] - 1, frames.shape[1] // slowfast_alpha
            ).long(),
        )
        frame_list = [slow_pathway, fast_pathway]
        return frame_list

transform =  ApplyTransformToKey(
    key="video",
    transform=Compose(
        [
            UniformTemporalSubsample(num_frames),
            Lambda(lambda x: x/255.0),
            NormalizeVideo(mean, std),
            ShortSideScale(
                size=side_size
            ),
            CenterCropVideo(crop_size),
            PackPathway()
        ]
    ),
)

# The duration of the input clip is also specific to the model.
clip_duration = (num_frames * sampling_rate)/frames_per_second
```

Run Inference

Download an example video

In [28]:
```
url_link = "https://dl.fbaipublicfiles.com/pytorchvideo/projects/archery.mp4"
video_path = 'archery.mp4'
try: urllib.URLopener().retrieve(url_link, video_path)
except: urllib.request.urlretrieve(url_link, video_path)
```

Load the video and transform it to the input format required by the model.

In [29]:
```
# Select the duration of the clip to load by specifying the start and end duratic
# The start_sec should correspond to where the action occurs in the video
start_sec = 0
end_sec = start_sec + clip_duration

# Initialize an EncodedVideo helper class and load the video
video = EncodedVideo.from_path(video_path)

# Load the desired clip
video_data = video.get_clip(start_sec=start_sec, end_sec=end_sec)

# Apply a transform to normalize the video input
video_data = transform(video_data)

# Move the inputs to the desired device
inputs = video_data["video"]
inputs = [i.to(device)[None, ...] for i in inputs]
```

Get Predictions

In [30]:
```
# Pass the input clip through the model
preds = model(inputs)

# Get the predicted classes
post_act = torch.nn.Softmax(dim=1)
preds = post_act(preds)
pred_classes = preds.topk(k=5).indices[0]

# Map the predicted classes to the label names
pred_class_names = [kinetics_id_to_classname[int(i)] for i in pred_classes]
print("Top 5 predicted labels: %s" % ", ".join(pred_class_names))
```

```
Top 5 predicted labels: archery, throwing axe, playing paintball, disc golfing,
riding or walking with horse
```