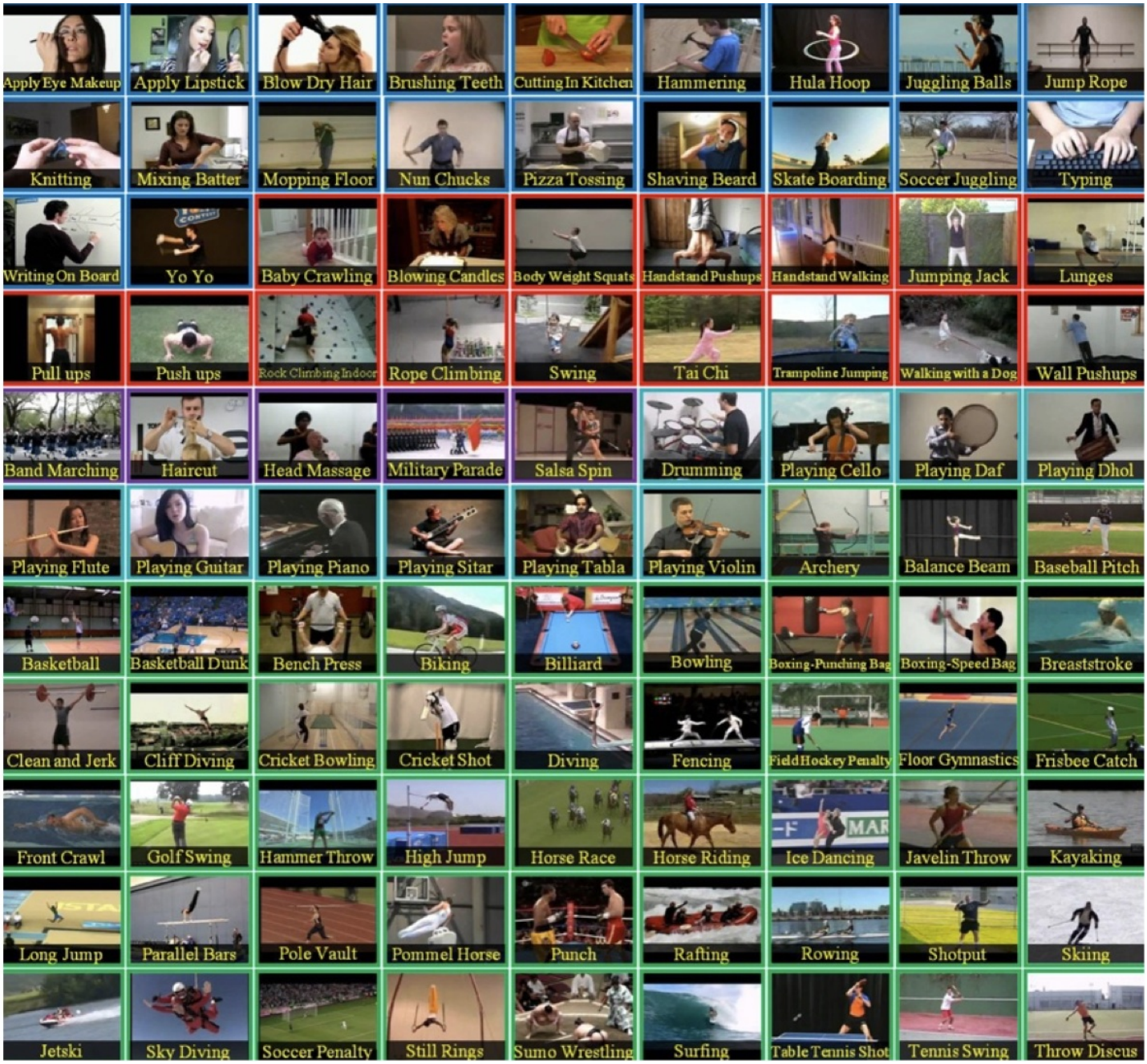
In [ ]:

```
import torch
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")
```

```
Using cpu device
```

Project 2 Action Recognition on UCF101

In this project, we will implement classical action recognition method C3D on UCF101 video dataset. The UCF101 video dataset is an action recognition dataset of realistic action videos, collected from YouTube, having 101 categories. The following picture illustrates some action categories in the dataset.



Before you start this project, please read the ICCV 2015 paper (Learning Spatiotemporal Features with 3D Convolutional Networks). You need to answer Q1, Q2, Q3, Q4 and Q5 in a report.

Q1

Prepare dataset. Visit the UCF101 official website and download the video dataset (https://www.crcv.ucf.edu/datasets/human-actions/ucf101/UCF101.rar). Unzip the dataset

in the “data” folder. Then read the dataset.py file in the “dataloaders” folder and answer how this code split videos into training, validation and testing dataset.

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
path_of_dataset=
```

In [1]:

```
# You can also using wget to download dataset into colab. Be careful not to exceed the co
lab daily limitation.
#!wget --no-check-certificate https://www.crcv.ucf.edu/datasets/human-actions/ucf101/UCF1
01.rar -P your_target_path
#unrar x -Y "your_target_path" "unzipp path"
```

dataloaders

In [ ]:

```
#you need to load datasetloader folder from google drive
import os
from sklearn.model_selection import train_test_split

import torch
import cv2
import numpy as np
from torch.utils.data import Dataset



class VideoDataset(Dataset):
    """A Dataset for a folder of videos. Expects the directory structure to be
    directory->[train/val/test]->[class labels]->[videos]. Initializes with a list
    of all file names, along with an array of labels, with label being automatically
    inferred from the respective folder names.

        Args:
            dataset (str): Name of dataset. Defaults to 'ucf101'.
            split (str): Determines which folder of the directory the dataset will read
from. Defaults to 'train'.
            clip_len (int): Determines how many frames are there in each clip. Defaults
to 16.
            preprocess (bool): Determines whether to preprocess dataset. Default is False
.
    """

    def __init__(self, dataset='ucf101', split='train', clip_len=16, preprocess=False):
        self.root_dir, self.output_dir = Path.db_dir(dataset)
        folder = os.path.join(self.output_dir, split)
        self.clip_len = clip_len
        self.split = split

        # The following three parameters are chosen as described in the paper section 4.1
        self.resize_height = 128
        self.resize_width = 171
        self.crop_size = 112

        if not self.check_integrity():
            raise RuntimeError('Dataset not found or corrupted.' +
                               ' You need to download it from official website.')

        if (not self.check_preprocess()) or preprocess:
            print('Preprocessing of {} dataset, this will take long, but it will be done
only once.'.format(dataset))
            self.preprocess()
        # Obtain all the filenames of files inside all the class folders
        # Going through each class folder one at a time
```

```
        self.fnames, labels = [], []
        for label in sorted(os.listdir(folder)):
            for fname in os.listdir(os.path.join(folder, label)):
                self.fnames.append(os.path.join(folder, label, fname))
                labels.append(label)

        assert len(labels) == len(self.fnames)
        print('Number of {} videos: {:d}'.format(split, len(self.fnames)))

        # Prepare a mapping between the label names (strings) and indices (ints)
        self.label2index = {label: index for index, label in enumerate(sorted(set(labels
))))}
        # Convert the list of label names into an array of label indices

        self.label_array = np.array([self.label2index[label] for label in labels], dtype
=int)

        if dataset == "ucf101":
            if not os.path.exists('dataloaders/ucf_labels.txt'):
                with open('dataloaders/ucf_labels.txt', 'w') as f:
                    for id, label in enumerate(sorted(self.label2index)):
                        f.writelines(str(id+1) + ' ' + label + '\n')

        elif dataset == 'hmdb51':
            if not os.path.exists('dataloaders/hmdb_labels.txt'):
                with open('dataloaders/hmdb_labels.txt', 'w') as f:
                    for id, label in enumerate(sorted(self.label2index)):
                        f.writelines(str(id+1) + ' ' + label + '\n')


    def __len__(self):
        return len(self.fnames)

    def __getitem__(self, index):
        # Loading and preprocessing.
        buffer = self.load_frames(self.fnames[index])
        buffer = self.crop(buffer, self.clip_len, self.crop_size)
        labels = np.array(self.label_array[index])

        if self.split == 'test':
            # Perform data augmentation
            buffer = self.randomflip(buffer)
        buffer = self.normalize(buffer)
        buffer = self.to_tensor(buffer)
        return buffer,labels
        # return torch.from_numpy(buffer), torch.from_numpy(labels)

    def check_integrity(self):
        if not os.path.exists(self.root_dir):
            return False
        else:
            return True

    def check_preprocess(self):
        # TODO: Check image size in output_dir
        if not os.path.exists(self.output_dir):
            return False
        elif not os.path.exists(os.path.join(self.output_dir, 'train')):
            return False

        for ii, video_class in enumerate(os.listdir(os.path.join(self.output_dir, 'train
'))):
            for video in os.listdir(os.path.join(self.output_dir, 'train', video_class))
:
                video_name = os.path.join(os.path.join(self.output_dir, 'train', video_c
lass, video),
                                          sorted(os.listdir(os.path.join(self.output_dir, 'tra
in', video_class, video)))[0])
                image = cv2.imread(video_name)
                if np.shape(image)[0] != 128 or np.shape(image)[1] != 171:
                    return False
                else:
```

```
                    break

            if ii == 10:
                break

        return True

    def preprocess(self):
         if not os.path.exists(self.output_dir):
            os.mkdir(self.output_dir)
            os.mkdir(os.path.join(self.output_dir, 'train'))
            os.mkdir(os.path.join(self.output_dir, 'val'))
            os.mkdir(os.path.join(self.output_dir, 'test'))

        # Split train/val/test sets
         for file in os.listdir(self.root_dir):
            file_path = os.path.join(self.root_dir, file)
            video_files = [name for name in os.listdir(file_path)]

            train_and_valid, test = train_test_split(video_files, test_size=0.2, random_
state=42)
            train, val = train_test_split(train_and_valid, test_size=0.2, random_state=4
2)

            train_dir = os.path.join(self.output_dir, 'train', file)
            val_dir = os.path.join(self.output_dir, 'val', file)
            test_dir = os.path.join(self.output_dir, 'test', file)

            if not os.path.exists(train_dir):

                train_dir = train_dir.replace("\ ", "\\")
                print("train " + train_dir)
                os.mkdir(train_dir)
            if not os.path.exists(val_dir):
                val_dir = val_dir.replace("\ ", "\\")
                os.mkdir(val_dir)
            if not os.path.exists(test_dir):
                test_dir = test_dir.replace("\ ", "\\ ")
                os.mkdir(test_dir)

            for video in train:
                self.process_video(video, file, train_dir)

            for video in val:
                self.process_video(video, file, val_dir)

            for video in test:
                self.process_video(video, file, test_dir)

         print('Preprocessing finished.')

    def process_video(self, video, action_name, save_dir):
        # Initialize a VideoCapture object to read video data into a numpy array
        video_filename = video.split('.')[0]
        if not os.path.exists(os.path.join(save_dir, video_filename)):
            os.mkdir(os.path.join(save_dir, video_filename))

        capture = cv2.VideoCapture(os.path.join(self.root_dir, action_name, video))

        frame_count = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
        frame_width = int(capture.get(cv2.CAP_PROP_FRAME_WIDTH))
        frame_height = int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT))

        # Make sure splited video has at least 16 frames
        EXTRACT_FREQUENCY = 4
        if frame_count // EXTRACT_FREQUENCY <= 16:
            EXTRACT_FREQUENCY -= 1
            if frame_count // EXTRACT_FREQUENCY <= 16:
                EXTRACT_FREQUENCY -= 1
                if frame_count // EXTRACT_FREQUENCY <= 16:
                    EXTRACT_FREQUENCY -= 1
```

```
        count = 0
        i = 0
        retaining = True

        while (count < frame_count and retaining):
            retaining, frame = capture.read()
            if frame is None:
                continue

            if count % EXTRACT_FREQUENCY == 0:
                if (frame_height != self.resize_height) or (frame_width != self.resize_w
idth):
                    frame = cv2.resize(frame, (self.resize_width, self.resize_height))
                cv2.imwrite(filename=os.path.join(save_dir, video_filename, '0000{}.jpg'
.format(str(i))), img=frame)
                i += 1
            count += 1

        # Release the VideoCapture once it is no longer needed
        capture.release()

    def randomflip(self, buffer):
        """Horizontally flip the given image and ground truth randomly with a probabilit
y of 0.5."""

        if np.random.random() < 0.5:
            for i, frame in enumerate(buffer):
                frame = cv2.flip(buffer[i], flipCode=1)
                buffer[i] = cv2.flip(frame, flipCode=1)

        return buffer


    def normalize(self, buffer):
        for i, frame in enumerate(buffer):
            frame -= np.array([[[90.0, 98.0, 102.0]]])
            buffer[i] = frame

        return buffer

    def to_tensor(self, buffer):
        return buffer.transpose((3, 0, 1, 2))

    def load_frames(self, file_dir):
        frames = sorted([os.path.join(file_dir, img) for img in os.listdir(file_dir)])
        frame_count = len(frames)
        buffer = np.empty((frame_count, self.resize_height, self.resize_width, 3), np.dt
ype('float32'))
        for i, frame_name in enumerate(frames):
            frame = np.array(cv2.imread(frame_name)).astype(np.float64)
            buffer[i] = frame

        return buffer

    def crop(self, buffer, clip_len, crop_size):
        # randomly select time index for temporal jittering
        time_index = np.random.randint(buffer.shape[0] - clip_len)

        # Randomly select start indices in order to crop the video
        height_index = np.random.randint(buffer.shape[1] - crop_size)
        width_index = np.random.randint(buffer.shape[2] - crop_size)

        # Crop and jitter the video using indexing. The spatial crop is performed on
        # the entire array, so each frame is cropped in the same location. The temporal
        # jitter takes place via the selection of consecutive frames
        buffer = buffer[time_index:time_index + clip_len,
                 height_index:height_index + crop_size,
                 width_index:width_index + crop_size, :]

        return buffer
    def getlabel2index(self):
        return self.label2index
```

```
if __name__ == "__main__":
    from torch.utils.data import DataLoader
    train_data = VideoDataset(dataset='ucf101', split='train', clip_len=16, preprocess=F
alse)
    train_loader = DataLoader(train_data, batch_size=1, shuffle=True, num_workers=0)

    for i, sample in enumerate(train_loader):
        inputs = sample[0]
        labels = sample[1]
        print(inputs.size())
        print(labels)

        if i == 1:
            break
```

Q2

Read the dataset.py file and answer how this code prepare the video data and the label.

Your Answer

Q3

Read the paper and reproduce the C3D network in C3D_model.py in network folder. (You need to implement the init and the forward function of C3D class.)

C3D.py

In [ ]:

```
import torch
import torch.nn as nn
from mypath import Path

class C3D(nn.Module):
    """
    The C3D network.
    """

    def __init__(self, num_classes, pretrained=False):
        super(C3D, self).__init__()
 # implement your code here



        self.__init_weight()

        if pretrained:
            self.__load_pretrained_weights()

    def forward(self, x):

        #implement your code here

        return logits

    def __load_pretrained_weights(self):
        """Initialiaze network."""
        corresp_name = {
                        # Conv1
                        "features.0.weight": "conv1.weight",
```

```
                        "features.0.bias": "conv1.bias",
                        # Conv2
                        "features.3.weight": "conv2.weight",
                        "features.3.bias": "conv2.bias",
                        # Conv3a
                        "features.6.weight": "conv3a.weight",
                        "features.6.bias": "conv3a.bias",
                        # Conv3b
                        "features.8.weight": "conv3b.weight",
                        "features.8.bias": "conv3b.bias",
                        # Conv4a
                        "features.11.weight": "conv4a.weight",
                        "features.11.bias": "conv4a.bias",
                        # Conv4b
                        "features.13.weight": "conv4b.weight",
                        "features.13.bias": "conv4b.bias",
                        # Conv5a
                        "features.16.weight": "conv5a.weight",
                        "features.16.bias": "conv5a.bias",
                         # Conv5b
                        "features.18.weight": "conv5b.weight",
                        "features.18.bias": "conv5b.bias",
                        # fc6
                        "classifier.0.weight": "fc6.weight",
                        "classifier.0.bias": "fc6.bias",
                        # fc7
                        "classifier.3.weight": "fc7.weight",
                        "classifier.3.bias": "fc7.bias",
                        }

        p_dict = torch.load(Path.model_dir())
        s_dict = self.state_dict()
        for name in p_dict:
            if name not in corresp_name:
                continue
            s_dict[corresp_name[name]] = p_dict[name]
        self.load_state_dict(s_dict)

    def __init_weight(self):
        for m in self.modules():
            if isinstance(m, nn.Conv3d):
                # n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                # m.weight.data.normal_(0, math.sqrt(2. / n))
                torch.nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, nn.BatchNorm3d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()

def get_1x_lr_params(model):
    """
    This generator returns all the parameters for conv and two fc layers of the net.
    """
    b = [model.conv1, model.conv2, model.conv3a, model.conv3b, model.conv4a, model.conv4
b,
         model.conv5a, model.conv5b, model.fc6, model.fc7]
    for i in range(len(b)):
        for k in b[i].parameters():
            if k.requires_grad:
                yield k

def get_10x_lr_params(model):
    """
    This generator returns all the parameters for the last fc layer of the net.
    """
    b = [model.fc8]
    for j in range(len(b)):
        for k in b[j].parameters():
            if k.requires_grad:
                yield k

if __name__ == "__main__":
    inputs = torch.rand(1, 3, 16, 112, 112)
```

```
    net = C3D(num_classes=101, pretrained=True)

    outputs = net.forward(inputs)
    print(outputs.size())
```

Q4

Read the train.py. Feel free to train the model if you have GPU resources (GPU memory: at least 2GB). If you do not have GPU resources, I have prepared a pretrained model (C3D-ucf101_epoch-19.pth.tar). If you need it please put it in the "run/run_0/model/" folder. Then explain the experiment details of train.py in your report. I spend about 1 hour on a RTX 3090 GPU with batch size=64. Please find the model files (c3d-init.pth and C3D-ucf101_epoch-19.pth.tar) on google drive and put c3d-init.pth in the "models" folder. (https://drive.google.com/drive/folders/0Alf6uQcTDj-LUk9PVA)

Q5

Q5

Test your model and provide one example of action recognition like demo1.gif in assets folder. I provide a test code, but it cannot generate the demo1.gif, add a function in your test code so that it can generate example of action recognition.

Please submit your jpynb and report to mycourse. Please do not submit your model file and dataset.

In [ ]:

```
```