```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.ap

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
%pwd
```

```
'/content'
```

```
cd '/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw5'
```

```
/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw5
```

```
%pwd
%ls -lrt
```

```
total 8455342
-rw------- 1 root root 8658247680 Oct 18 17:35 UCF101_images.tar
drwx------ 2 root root       4096 Nov 20 11:13 annos/
-rw------- 1 root root      17664 Nov 21 03:49 CSE527_HW5_fall19.ipynb
```

```
!wget 'http://vision.cs.stonybrook.edu/~yangwang/public/UCF101_images.tar'
```

```
--2019-11-21 03:39:18--  http://vision.cs.stonybrook.edu/~yangwang/public/UCF101_images.tar
Resolving vision.cs.stonybrook.edu (vision.cs.stonybrook.edu)... 130.245.4.232
Connecting to vision.cs.stonybrook.edu (vision.cs.stonybrook.edu)|130.245.4.232|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8658247680 (8.1G) [application/x-tar]
Saving to: ‘UCF101_images.tar.1’

UCF101_images.tar.1 100%[===================>]   8.06G  31.9MB/s    in 4m 30s

2019-11-21 03:43:48 (30.6 MB/s) - ‘UCF101_images.tar.1’ saved [8658247680/8658247680]
```

```
!tar -xkf './UCF101_images.tar' 2>/dev/null
```

```
cd '/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw5/'
```

```
/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw5
```

```
%ls -lrt
```

```
total 8455446
drwx------ 13322 root root       4096 May 31  2017 images/
-rw-------     1 root root 8658247680 Oct 18 17:35 UCF101_images.tar
drwx------     2 root root       4096 Nov 20 11:13 annos/
-rw-------     1 root root      66439 Nov 21 08:20 train_video_label_df.pkl
-rw-------     1 root root      26417 Nov 21 08:20 test_video_label_df.pkl
-rw-------     1 root root      26921 Nov 21 08:31 Paratkar_Shreyash_112673930_hw5.ipynb
```

Action Recognition @ UCF101

Due date: 11:59 pm on Nov. 19, 2019 (Tuesday)

Description

In this homework, you will be doing action recognition using Recurrent Neural Network (RNN), (Long-Short Term Memory) LSTM in particular. You will be given a dataset called UCF101, which consists of 101 different actions/classes and for each action, there will be 145 samples. We tagged each sample into either training or testing. Each sample is supposed to be a short video, but we sampled 25 frames from each videos to reduce the amount of data. Consequently, a training sample is an image tuple that forms a 3D volume with one dimension encoding *temporal correlation* between frames and a label indicating what action it is.

To tackle this problem, we aim to build a neural network that can not only capture spatial information of each frame but also temporal information between frames. Fortunately, you don't have to do this on your own. RNN — a type of neural network designed to deal with time-series data — is right here for you to use. In particular, you will be using LSTM for this task.

Instead of training an end-to-end neural network from scratch whose computation is prohibitively expensive, we divide this into two steps: feature extraction and modelling. Below are the things you need to implement for this homework:

- **{35 pts} Feature extraction**. Use any of the pre-trained models to extract features from each frame. Specifically, we recommend not to use the activations of the last layer as the features tend to be task specific towards the end of the network. **hints**:
    - A good starting point would be to use a pre-trained VGG16 network, we suggest first fully connected layer `torchvision.models.vgg16` (4096 dim) as features of each video frame. This will result into a 4096x25 matrix for each video.
    - Normalize your images using `torchvision.transforms`

    ```
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    prep = transforms.Compose([ transforms.ToTensor(), normalize ])
    prep(img)
    The mean and std. mentioned above is specific to Imagenet data
    ```

    More details of image preprocessing in PyTorch can be found at http://pytorch.org/tutorials/beginner/data_loading_tutorial.html
- **{35 pts} Modelling**. With the extracted features, build an LSTM network which takes a **dx25** sample as input (where **d** is the dimension of the extracted feature for each frame), and outputs the action label of that sample.
- **{20 pts} Evaluation**. After training your network, you need to evaluate your model with the testing data by computing the prediction accuracy **(5 points)**. The baseline test accuracy for this data is 75%, and **10 points** out of 20 is for achieving test accuracy greater than the baseline. Moreover, you need to compare **(5 points)** the result of your network with that of support vector machine (SVM) (stacking the **dx25** feature matrix to a long vector and train a SVM).
- **{10 pts} Report**. Details regarding the report can be found in the submission section below.

Notice that the size of the raw images is 256x340, whereas your pre-trained model might take **nxn** images as inputs. To solve this problem, instead of resizing the images which unfavorably changes the spatial ratio, we take a better solution: Cropping five **nxn** images, one at the image center and four at the corners and compute the **d**-dim features for each of them, and average these five **d**-dim feature to get a final feature representation for the raw image. For example, VGG takes 224x224 images as inputs, so we take the five 224x224 croppings of the image, compute 4096-dim VGG features for each of them, and then take the mean of these five 4096-dim vectors to be the representation of the image.

In order to save you computational time, you need to do the classification task only for **the first 25** classes of the whole dataset. The same applies to those who have access to GPUs. **Bonus 10 points for running and reporting on the entire 101 classes.**

Dataset

Download **dataset** at UCF101(Image data for each video) and the **annos folder** which has the video labels and the label to class name mapping is included in the assignment folder uploaded.

UCF101 dataset contains 101 actions and 13,320 videos in total.

- `annos/actions.txt`
  - lists all the actions (`ApplyEyeMakeup` , .., `YoYo`)
- `annots/videos_labels_subsets.txt`
  - lists all the videos (`v_000001` , .., `v_013320`)
  - labels (`1` , .., `101`)
  - subsets (`1` for train, `2` for test)
- `images/`
  - each folder represents a video
  - the video/folder name to class mapping can be found using `annots/videos_labels_subsets.txt` , for e.g. `v_000001` belongs to class 1 i.e. `ApplyEyeMakeup`
  - each video folder contains 25 frames

Some Tutorials

- Good materials for understanding RNN and LSTM
  - http://blog.echen.me
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Implementing RNN and LSTM with PyTorch
  - LSTM with PyTorch
  - RNN with PyTorch

Problem 1. Feature extraction

```
# \*write your codes for feature extraction (You can use multiple cells, this is just a place holder)
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
import torchvision.models as models
from __future__ import print_function, division
import torch
import pandas as pd
from skimage import io, transform
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
import pickle
import cv2
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
# Ignore warnings
import warnings
import time
warnings.filterwarnings("ignore")

plt.ion()   # interactive mode
```

```
vgg16 = models.vgg16(pretrained=True)
vgg16.classifier = vgg16.classifier[:2]
vgg16
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:21<00:00, 25.8MB/s]
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
  )
)
```

```
videos_labels_subsets = open("annos/videos_labels_subsets.txt", "r")
```

```
videos_labels_subsets = open('annos/videos_labels_subsets.txt', 'r')
dic = {}
train_list = []
test_list = []
cnt=0
for l in videos_labels_subsets:
  lst = l[:-1].split('\t')
  if int(lst[1]) < 21 or int(lst[1]) > 25:
    continue
  dic['v_num'] = lst[0]
  dic['class'] = lst[1]
  if lst[2]=='1':
    train_list.append(dic)
  else:
    test_list.append(dic)
  dic = {}
train_video_label_df = pd.DataFrame(train_list)
test_video_label_df = pd.DataFrame(test_list)
file = open('train_video_label_df.pkl','wb')
pickle.dump(train_video_label_df, file)
file.close()
file = open('test_video_label_df.pkl','wb')
pickle.dump(test_video_label_df, file)
file.close()
print(len(train_list))
print(len(test_list))
```

```
476
190
```

```
file = open('train_video_label_df.pkl', 'rb')
train_video_label_df = pickle.load(file)
file.close()
file = open('test_video_label_df.pkl', 'rb')
test_video_label_df = pickle.load(file)
file.close()
```

```
import os
videos_labels_subsets2 = open("annos/videos_labels_subsets.txt", "r")
img_count = 0
for l in videos_labels_subsets2:
```

```
  lst = l[:-1].split('\t')
  if int(lst[1]) > 25:
    break
  path, dirs, files = next(os.walk('/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_1126739
  img_count = img_count + len(files)
img_count
```

```
root_dir = '/content/gdrive/My Drive/Y2019Fall/CSE-527-Intro-To-Computer-Vision/Paratkar_Shreyash_112673930_hw5/images/'
# image = cv2.imread(root_dir+'v_000001/i_0001.jpg')
# # print(torch.tensor(image).shape)
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
prep = transforms.Compose([ transforms.ToTensor(), normalize ])
# image = prep(image)
# print(image.shape)
# img_tp_lt = image[0:3, :224, :224]
# print(img_tp_lt.shape)
# img_bt_lt = image[0:3, 32:, :224]
# print(img_bt_lt.shape)
# img_bt_rt = image[0:3, 32:, 116:]
# print(img_bt_rt.shape)
# img_tp_rt = image[0:3, :224, 116:]
# print(img_tp_rt.shape)
# img_cn = image[0:3, 16:240, 58:282]
# print(img_cn.shape)
# labels = [1] * 125
# labels = torch.Tensor(np.array(labels))
# print(labels)
```

```
print(torch.cuda.memory_cached()-torch.cuda.memory_allocated())
```

```
0
```

```
class UCF101ImageDataset(Dataset):
    """UCF101 Actions dataset."""

    def __init__(self, pickle_file, root_dir, transform=transforms.Compose([
                                               transforms.ToTensor(),
                                               transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
                                           ])):
        """
```

```
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        file = open(pickle_file, 'rb')
        self.actions_frame = pickle.load(file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        action_num = self.actions_frame.iloc[idx, 1]
        image_transforms = []
        for i in range(1, 26):
          img_name = ''
          if i<10:
            img_name = 'i_000'+str(i)+'.jpg'
          else:
            img_name = 'i_00'+str(i)+'.jpg'
          parent_img_name = self.root_dir+self.actions_frame.iloc[idx, 0]+ '/'+ img_name
          image = cv2.imread(parent_img_name)
          # print(parent_img_name)
          image = prep(image)
          # print(image.shape)
          image_transforms.append(image[0:3, :224, :224])
          # print(img_tp_lt.shape)
          image_transforms.append(image[0:3, 32:, :224])
          # print(img_bt_lt.shape)
          image_transforms.append(image[0:3, 32:, 116:])
          # print(img_bt_rt.shape)
          image_transforms.append(image[0:3, :224, 116:])
          # print(img_tp_rt.shape)
          image_transforms.append(image[0:3, 16:240, 58:282])
          # print(img_cn.shape)
        label = int(action_num)
        # print(len(image_transforms))
        # print(torch.stack(image_transforms).shape)
```

```
        # print(torch.stack(image_transforms).shape)
        sample = {'images':torch.stack(image_transforms), 'action':label}
        return sample
train_dataset_func = UCF101ImageDataset(pickle_file='train_video_label_df.pkl', root_dir=root_dir)
test_dataset_func = UCF101ImageDataset(pickle_file='test_video_label_df.pkl', root_dir=root_dir)
train_dataloader = DataLoader(train_dataset_func, batch_size=1, shuffle=False, num_workers=4)
test_dataloader = DataLoader(test_dataset_func, batch_size=1, shuffle=False, num_workers=4)
```

```
# torch.cuda.empty_cache()
!nvidia-smi
```

```
Fri Nov 22 07:59:51 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 430.50       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   35C    P8    26W / 149W |     11MiB / 11441MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
import time
model = vgg16
model = model.cuda()
count = 0
train_features_data = []
labels = []
model.eval()
start_time = time.time()
with torch.no_grad():
  for video_num, sample in enumerate(test_dataloader):
    print("Video taken: ", video_num)
    print("Time since start ", round(time.time() - start_time), " seconds")
    video_images_tensor = sample['images'][0].cuda()
```

```
    label = sample['action'][0].cuda()

    vgg16output_125 = model(video_images_tensor)

    del video_images_tensor

    for i in range(0, 125, 5):
      tensor = vgg16output_125[i]
      # print("First tensor: ", tensor)
      for j in range(1,5):
        tensor+=vgg16output_125[i+j]
      # print("Mean tensor: ", tensor)
      train_features_data.append(tensor/5)
      labels.append(label)
      del tensor
    del vgg16output_125, label
print("Length of training features: ", len(train_features_data))
print("Length labels: ", len(labels))
```

```
Video taken:  0
Time since start  22  seconds
Video taken:  1
Time since start  24  seconds
Video taken:  2
Time since start  25  seconds
Video taken:  3
Time since start  26  seconds
Video taken:  4
Time since start  28  seconds
Video taken:  5
Time since start  29  seconds
Video taken:  6
Time since start  31  seconds
Video taken:  7
Time since start  32  seconds
Video taken:  8
Time since start  34  seconds
Video taken:  9
Time since start  36  seconds
Video taken:  10
Time since start  37  seconds
Video taken:  11
Time since start  38  seconds
Video taken:  12
Time since start  40  seconds
Video taken:  13
Time since start  41  seconds
Video taken:  14
Time since start  43  seconds
Video taken:  15
Time since start  44  seconds
Video taken:  16
Time since start  46  seconds
Video taken:  17
Time since start  47  seconds
Video taken:  18
Time since start  49  seconds
Video taken:  19
Time since start  50  seconds
Video taken:  20
Time since start  53  seconds
Video taken:  21
Time since start  54  seconds
Video taken:  22
Time since start  56  seconds
```

```
Time since start  56  seconds
Video taken:  23
Time since start  57  seconds
Video taken:  24
Time since start  59  seconds
Video taken:  25
Time since start  61  seconds
Video taken:  26
Time since start  62  seconds
Video taken:  27
Time since start  64  seconds
Video taken:  28
Time since start  65  seconds
Video taken:  29
Time since start  67  seconds
Video taken:  30
Time since start  68  seconds
Video taken:  31
Time since start  70  seconds
Video taken:  32
Time since start  71  seconds
Video taken:  33
Time since start  73  seconds
Video taken:  34
Time since start  75  seconds
Video taken:  35
Time since start  76  seconds
Video taken:  36
Time since start  79  seconds
Video taken:  37
Time since start  80  seconds
Video taken:  38
Time since start  82  seconds
Video taken:  39
Time since start  83  seconds
Video taken:  40
Time since start  85  seconds
Video taken:  41
Time since start  87  seconds
Video taken:  42
Time since start  88  seconds
Video taken:  43
Time since start  89  seconds
Video taken:  44
Time since start  91  seconds
Video taken:  45
```

```
Time since start  95  seconds
Video taken:  46
Time since start  97  seconds
Video taken:  47
Time since start  98  seconds
Video taken:  48
Time since start  99  seconds
Video taken:  49
Time since start  101  seconds
Video taken:  50
Time since start  103  seconds
Video taken:  51
Time since start  104  seconds
Video taken:  52
Time since start  105  seconds
Video taken:  53
Time since start  108  seconds
Video taken:  54
Time since start  109  seconds
Video taken:  55
Time since start  110  seconds
Video taken:  56
Time since start  111  seconds
Video taken:  57
Time since start  113  seconds
Video taken:  58
Time since start  115  seconds
Video taken:  59
Time since start  116  seconds
Video taken:  60
Time since start  117  seconds
Video taken:  61
Time since start  120  seconds
Video taken:  62
Time since start  121  seconds
Video taken:  63
Time since start  122  seconds
Video taken:  64
Time since start  123  seconds
Video taken:  65
Time since start  126  seconds
Video taken:  66
Time since start  127  seconds
Video taken:  67
Time since start  128  seconds
Video taken:  68
```

```
Time since start  130  seconds
Video taken:  69
Time since start  132  seconds
Video taken:  70
Time since start  133  seconds
Video taken:  71
Time since start  135  seconds
Video taken:  72
Time since start  136  seconds
Video taken:  73
Time since start  138  seconds
Video taken:  74
Time since start  140  seconds
Video taken:  75
Time since start  141  seconds
Video taken:  76
Time since start  142  seconds
Video taken:  77
Time since start  145  seconds
Video taken:  78
Time since start  146  seconds
Video taken:  79
Time since start  147  seconds
Video taken:  80
Time since start  148  seconds
Video taken:  81
Time since start  152  seconds
Video taken:  82
Time since start  153  seconds
Video taken:  83
Time since start  154  seconds
Video taken:  84
Time since start  155  seconds
Video taken:  85
Time since start  158  seconds
Video taken:  86
Time since start  159  seconds
Video taken:  87
Time since start  160  seconds
Video taken:  88
Time since start  161  seconds
Video taken:  89
Time since start  164  seconds
Video taken:  90
Time since start  165  seconds
Video taken:  91
```

```
Video taken:  91
Time since start  166  seconds
Video taken:  92
Time since start  167  seconds
Video taken:  93
Time since start  169  seconds
Video taken:  94
Time since start  171  seconds
Video taken:  95
Time since start  172  seconds
Video taken:  96
Time since start  173  seconds
Video taken:  97
Time since start  175  seconds
Video taken:  98
Time since start  177  seconds
Video taken:  99
Time since start  178  seconds
Video taken:  100
Time since start  179  seconds
Video taken:  101
Time since start  182  seconds
Video taken:  102
Time since start  183  seconds
Video taken:  103
Time since start  184  seconds
Video taken:  104
Time since start  185  seconds
Video taken:  105
Time since start  188  seconds
Video taken:  106
Time since start  189  seconds
Video taken:  107
Time since start  190  seconds
Video taken:  108
Time since start  191  seconds
Video taken:  109
Time since start  194  seconds
Video taken:  110
Time since start  195  seconds
Video taken:  111
Time since start  196  seconds
Video taken:  112
Time since start  198  seconds
Video taken:  113
Time since start  201  seconds
Video taken:  114
```

```
Video taken:  114
Time since start  202  seconds
Video taken:  115
Time since start  203  seconds
Video taken:  116
Time since start  204  seconds
Video taken:  117
Time since start  207  seconds
Video taken:  118
Time since start  208  seconds
Video taken:  119
Time since start  209  seconds
Video taken:  120
Time since start  210  seconds
Video taken:  121
Time since start  213  seconds
Video taken:  122
Time since start  214  seconds
Video taken:  123
Time since start  215  seconds
Video taken:  124
Time since start  216  seconds
Video taken:  125
Time since start  219  seconds
Video taken:  126
Time since start  220  seconds
Video taken:  127
Time since start  221  seconds
Video taken:  128
Time since start  222  seconds
Video taken:  129
Time since start  224  seconds
Video taken:  130
Time since start  226  seconds
Video taken:  131
Time since start  227  seconds
Video taken:  132
Time since start  228  seconds
Video taken:  133
Time since start  230  seconds
Video taken:  134
Time since start  232  seconds
Video taken:  135
Time since start  233  seconds
Video taken:  136
Time since start  234  seconds
```

```
Video taken:  137
Time since start  237  seconds
Video taken:  138
Time since start  238  seconds
Video taken:  139
Time since start  239  seconds
Video taken:  140
Time since start  240  seconds
Video taken:  141
Time since start  243  seconds
Video taken:  142
Time since start  244  seconds
Video taken:  143
Time since start  245  seconds
Video taken:  144
Time since start  246  seconds
Video taken:  145
Time since start  250  seconds
Video taken:  146
Time since start  251  seconds
Video taken:  147
Time since start  252  seconds
Video taken:  148
Time since start  253  seconds
Video taken:  149
Time since start  256  seconds
Video taken:  150
Time since start  257  seconds
Video taken:  151
Time since start  258  seconds
Video taken:  152
Time since start  259  seconds
Video taken:  153
Time since start  263  seconds
Video taken:  154
Time since start  265  seconds
Video taken:  155
Time since start  266  seconds
Video taken:  156
Time since start  267  seconds
Video taken:  157
Time since start  270  seconds
Video taken:  158
Time since start  271  seconds
Video taken:  159
Time since start  272  seconds
```

```
Video taken:  160
Time since start  273  seconds
Video taken:  161
Time since start  276  seconds
Video taken:  162
Time since start  277  seconds
Video taken:  163
Time since start  279  seconds
Video taken:  164
Time since start  280  seconds
Video taken:  165
Time since start  285  seconds
Video taken:  166
Time since start  287  seconds
Video taken:  167
Time since start  288  seconds
Video taken:  168
Time since start  289  seconds
Video taken:  169
Time since start  292  seconds
Video taken:  170
Time since start  293  seconds
Video taken:  171
Time since start  294  seconds
Video taken:  172
Time since start  295  seconds
Video taken:  173
Time since start  298  seconds
Video taken:  174
Time since start  299  seconds
Video taken:  175
Time since start  301  seconds
Video taken:  176
Time since start  302  seconds
Video taken:  177
Time since start  305  seconds
Video taken:  178
Time since start  306  seconds
Video taken:  179
Time since start  307  seconds
Video taken:  180
Time since start  308  seconds
Video taken:  181
Time since start  312  seconds
Video taken:  182
Time since start  313  seconds
```

```
Video taken:  183
Time since start  314  seconds
Video taken:  184
Time since start  316  seconds
Video taken:  185
Time since start  319  seconds
Video taken:  186
Time since start  320  seconds
Video taken:  187
Time since start  321  seconds
Video taken:  188
Time since start  322  seconds
Video taken:  189
Time since start  326  seconds
Length of training features:  4750
Length labels:  4750
```

```
# file = open('test_features_data_class_21_25.pkl','wb')
# pickle.dump(train_features_data, file)
# file.close()
# file = open('test_labels_class_21_25.pkl','wb')
# pickle.dump(labels, file)
# file.close()
print("Length of training features: ", len(train_features_data))
print("Length labels: ", len(labels))
# file = open('train_features_data_class_21_25.pkl','wb')
# pickle.dump(train_features_data, file)
# file.close()
# file = open('labels_class_21_25.pkl','wb')
# pickle.dump(labels, file)
file.close()
```

```
Length of training features:  4750
Length labels:  4750
```

```
# file = open('test_features_data_class_11_15.pkl', 'rb')
# test_features_data_class_11_15 = pickle.load(file)
# print(len(test_features_data_class_11_15))
# file.close()
# file = open('test_labels_class_11_15.pkl', 'rb')
# test_labels_class_11_15 = pickle.load(file)
```

```
# print(len(test_labels_class_11_15))
# file.close()
# file = open('train_features_data_class_1_5.pkl', 'rb')
# train_features_data_class_0_5 = pickle.load(file)
# print(len(train_features_data_class_0_5))
# file.close()
# file = open('train_features_data_class_1_5.pkl', 'rb')
# labels_class_0_5 = pickle.load(file)
# print(len(labels_class_0_5))
# file.close()
```

```
11525
11525
```

```
train_pkls = ['train_features_data_class_1_5.pkl', 'train_features_data_class_6_10.pkl', 'train_features_data_class_11_15.pkl', '
train_label_pkls = ['labels_class_1_5.pkl', 'labels_class_6_10.pkl', 'labels_class_11_15.pkl', 'labels_class_16_20.pkl', 'labels_
test_pkls = ['test_features_data_class_1_5.pkl', 'test_features_data_class_6_10.pkl',  'test_features_data_class_11_15.pkl', 'tes
test_label_pkls = ['test_labels_class_1_5.pkl', 'test_labels_class_6_10.pkl', 'test_labels_class_11_15.pkl', 'test_labels_class_1
train_list = []
train_labels = []
test_list = []
test_labels = []
for i in range(0,5):
  train_list.extend(pickle.load(open(train_pkls[i], 'rb')))
  train_labels.extend(pickle.load(open(train_label_pkls[i], 'rb')))
  test_list.extend(pickle.load(open(test_pkls[i], 'rb')))
  test_labels.extend(pickle.load(open(test_label_pkls[i], 'rb')))
print(len(train_list))
print(len(train_labels))
print(len(test_list))
print(len(test_labels))
```

```
60225
60225
23775
23775
```

```
train_data_final = torch.stack(train_list)
print("Training data size:\t", train_data_final.size(), "\t type: ", type(train_data_final), "\t\t element: ", (train_data_final[
train_labels_final = torch.stack(train_labels)
print("Training labels size:\t", train_labels_final.size(), "\t\t type: ", type(train_labels_final), "\t\t element: ", (train_lab
```

```
print("Training labels size:\t", train_labels_final.size(), "\t\t type: ", type(train_labels_final), "\t\t element: ", (train_lab
test_data_final = torch.stack(test_list)
print("Testing data size:\t", test_data_final.size(), "\t type: ", type(test_data_final), "\t\t element: ", (test_data_final[0]))
test_labels_final = torch.stack(test_labels)
print("Testing labels size:\t", test_labels_final.size(), "\t\t type: ", type(test_labels_final), "\t\t element: ", (test_labels_
```

```
Training data size:      torch.Size([60225, 4096])      type:  <class 'torch.Tensor'>          element:  tensor([0.0542, 1.0759
Training labels size:    torch.Size([60225])            type:  <class 'torch.Tensor'>          element:  tensor(1, device='cuda
Testing data size:       torch.Size([23775, 4096])      type:  <class 'torch.Tensor'>          element:  tensor([0.0694, 0.0000
Testing labels size:     torch.Size([23775])            type:  <class 'torch.Tensor'>          element:  tensor(1, device='cuda
```

Problem 2. Modelling

Print the size of your training and test data

```
# Don't hardcode the shape of train and test data
print('Shape of training data is :', )
print("Training data size:\t", train_data_final.size(), "\t shape: ", train_data_final.shape, "\t type: ", type(train_data_final)
print("Training labels size:\t", train_labels_final.size(), "\t\t shape: ", train_labels_final.shape, "\t\t type: ", type(train_l
print('Shape of test/validation data is :', )
print("Testing data size:\t", test_data_final.size(), "\t shape: ", test_data_final.shape, "\t type: ", type(test_data_final), "\
print("Testing labels size:\t", test_labels_final.size(), "\t\t shape: ", test_labels_final.shape, "\t\t type: ", type(test_label
```

```
Shape of training data is :
Training data size:      torch.Size([60225, 4096])      shape:  torch.Size([60225, 4096])      type:  <class 'torch.Tensor'>
Training labels size:    torch.Size([60225])            shape:  torch.Size([60225])            type:  <class 'torch.Tensor'>
Shape of test/validation data is :
Testing data size:       torch.Size([23775, 4096])      shape:  torch.Size([23775, 4096])      type:  <class 'torch.Tensor'>
Testing labels size:     torch.Size([23775])            shape:  torch.Size([23775])            type:  <class 'torch.Tensor'>
```

```
file = open('train_data_final.pkl','wb')
pickle.dump(train_data_final, file)
file.close()
file = open('train_labels_final.pkl','wb')
pickle.dump(train_labels_final, file)
file.close()
file = open('test_data_final.pkl','wb')
pickle.dump(test_data_final, file)
```

```
file.close()
file = open('test_labels_final.pkl','wb')
pickle.dump(test_labels_final, file)
file.close()
```

Load final data from here

```
train_data_final = pickle.load(open('train_data_final.pkl', 'rb'))
train_labels_final = pickle.load(open('train_labels_final.pkl', 'rb'))
test_data_final = pickle.load(open('test_data_final.pkl', 'rb'))
test_labels_final = pickle.load(open('test_labels_final.pkl', 'rb'))
```

```
print("Training data size:\t", train_data_final.size(), "\t shape: ", train_data_final.shape, "\t type: ", type(train_data_final)
print("Training labels size:\t", train_labels_final.size(), "\t\t shape: ", train_labels_final.shape, "\t\t type: ", type(train_l
print("Testing data size:\t", test_data_final.size(), "\t shape: ", test_data_final.shape, "\t type: ", type(test_data_final), "\
print("Testing labels size:\t", test_labels_final.size(), "\t\t shape: ", test_labels_final.shape, "\t\t type: ", type(test_label
```

```
Training data size:      torch.Size([60225, 4096])       shape:  torch.Size([60225, 4096])       type:  <class 'torch.Tensor'>
Training labels size:    torch.Size([60225])             shape:  torch.Size([60225])             type:  <class 'torch.Tensor'>
Testing data size:       torch.Size([23775, 4096])       shape:  torch.Size([23775, 4096])       type:  <class 'torch.Tensor'>
Testing labels size:     torch.Size([23775])             shape:  torch.Size([23775])             type:  <class 'torch.Tensor'>
```

```
train_data_final_split = torch.stack(torch.split(train_data_final, 25))
print(train_data_final_split.shape)
train_labels_final_split = torch.stack(torch.split(train_labels_final, 25))
train_lab = []
for i in range(train_labels_final_split.shape[0]):
  # print(type(torch.max(torch.stack(torch.split(train_labels_final_split[i], 1)))))
  train_lab.append(torch.max(torch.stack(torch.split(train_labels_final_split[i], 1))))
train_labels_final_split = torch.tensor(train_lab)
print(train_labels_final_split.shape)
test_data_final_split = torch.stack(torch.split(test_data_final, 25))
print(test_data_final_split.shape)
test_labels_final_split = torch.stack(torch.split(test_labels_final, 25))
test_lab = []
for i in range(test_labels_final_split.shape[0]):
  # print(type(torch.max(torch.stack(torch.split(train_labels_final_split[i], 1)))))
  test_lab.append(torch.max(torch.stack(torch.split(test_labels_final_split[i], 1))))
```

```
test_labels_final_split = torch.tensor(test_lab)
print(test_labels_final_split.shape)
del train_lab, train_data_final, train_labels_final, test_lab, test_data_final, test_labels_final
```

```
torch.Size([2409, 25, 4096])
torch.Size([2409])
torch.Size([951, 25, 4096])
torch.Size([951])
```

```
import random
class LSTM(nn.Module):

    def __init__(self, input_dim, hidden_dim, batch_size, output_dim=1, num_layers=2):
        super(LSTM, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.batch_size = batch_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(self.input_dim, self.hidden_dim, self.num_layers)
        self.linear = nn.Linear(self.hidden_dim, output_dim)

    def init_hidden(self):
        return (torch.zeros(self.num_layers, self.batch_size, self.hidden_dim), torch.zeros(self.num_layers, self.batch_size, sel

    def forward(self, input):
        lstm_out, self.hidden = self.lstm(input.view(len(input), self.batch_size, -1))
        y_pred = self.linear(lstm_out[-1].view(self.batch_size, -1))
        return y_pred.view(-1)

def trainLSTM(model, video_features, video_labels, epochs, learning_rate,optimizer=None):
  device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
  optimizer = optim.Adam(model.parameters(), lr=learning_rate)
  model = model.cuda(device)
  loss_func = nn.CrossEntropyLoss()
  start_time = time.time()
  for ep in range(epochs):
      data = list(zip(video_features, video_labels))
      random.shuffle(data)
      video_features, video_labels = zip(*data)
      correct_count = 0
      total_count = 0
```

```
      for i in range(len(video_features)):
          video_feature = video_features[i]
          label_feature = video_labels[i].type(torch.LongTensor)
          label_feature = label_feature.view(1) - 1
          model.zero_grad()
          model.hidden = model.init_hidden()
          preds = model(video_feature.cuda(device))
          preds = preds.view(1, preds.shape[0])
          loss = loss_func(preds, label_feature.cuda(device))
          loss.backward()
          optimizer.step()
          pred_indices, pred_vals = torch.max(preds.data, 1)
          total_count = total_count + label_feature.size(0)
          correct_count = correct_count + (pred_vals == label_feature.cuda(device)).sum().item()
      torch.cuda.empty_cache()
      print('Epoch: ', ep + 1, ' = ' , round((correct_count / total_count) * 100, 3), '%')

  print('\nTraining time = ', round(time.time() - start_time), ' seconds')
  return (correct_count / total_count) * 100
model = LSTM(input_dim=4096, hidden_dim=1024, num_layers=2, batch_size=1, output_dim = 25)
print('Training accuracies per epoch:\n')
train_accuracy = trainLSTM(model, train_data_final_split, train_labels_final_split, 5, 0.00001)
print('LSTM Training Accuracy = ', train_accuracy)
```

```
Training accuracies per epoch:

Epoch:  1  =  74.72 %
Epoch:  2  =  99.045 %
Epoch:  3  =  99.751 %
Epoch:  4  =  99.917 %
Epoch:  5  =  100.0 %

Training time =  228  seconds
LSTM Training Accuracy =  100.0
```

```
import time
def test_model(model, video_features, video_labels):
  device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
  start_time = time.time()
  model.eval()
  with torch.no_grad():
      correct_count = 0
```

```
      correct_count = 0
      total_count = 0
      for i in range(video_features.shape[0]):
          video_feature = video_features[i]
          label_feature = video_labels[i].type(torch.LongTensor)
          label_feature = label_feature.view(1) - 1

          preds = model(video_feature.cuda(device))
          preds = preds.view(1, preds.shape[0])
          _, pred_vals = torch.max(preds.data, 1)
          total_count = total_count + label_feature.size(0)
          correct_count = correct_count + (pred_vals == label_feature.cuda(device)).sum().item()

      print('Test accuracy = ' , round((correct_count / total_count) * 100, 3), '%')

  print('Testing time: ', round(time.time() - start_time), ' seconds')
  return (correct_count / total_count) * 100
test_accuracy = test_model(model, test_data_final_split, test_labels_final_split)
print('LSTM Testing Accuracy = ', test_accuracy)
```

```
Test accuracy =  80.967 %
Testing time:  4  seconds
LSTM Testing Accuracy =  80.96740273396425
```

Reference:

https://www.jessicayung.com/lstms-for-time-series-in-pytorch/

```
train_for_linearSVC = train_data_final_split.view(train_data_final_split.shape[0], train_data_final_split.shape[1]* train_data_fi
train_labels_for_linearSVC = train_labels_final_split.cpu().detach().numpy()
test_for_linearSVC = test_data_final_split.view(test_data_final_split.shape[0], test_data_final_split.shape[1]* test_data_final_s
test_labels_for_linearSVC = test_labels_final_split.cpu().detach().numpy()
```

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
import time
start_time = time.time()
linearSVCClassifier = LinearSVC(C=0.0001, multi_class="ovr")
linearSVCClassifier.fit(train_for_linearSVC, train_labels_for_linearSVC)
pred = linearSVCClassifier.predict(test_for_linearSVC)
test_accuracy_SVC = accuracy_score(pred, test_labels_for_linearSVC)*100
```

```
test_accuracy_SVC = accuracy_score(pred, test_labels_for_linearSVC)*100
print('Test accuracy: ', test_accuracy_SVC, '%')
print('Testing time: ', round(time.time() - start_time), ' seconds')
```

```
Test accuracy:  86.01472134595163 %
Testing time:  195  seconds
```

```
linearSVCClassifier = LinearSVC(C=0.0001, multi_class="ovr")
linearSVCClassifier.fit(train_for_linearSVC, train_labels_for_linearSVC)
pred = linearSVCClassifier.predict(train_for_linearSVC)
train_accuracy_SVC = accuracy_score(pred, train_labels_for_linearSVC)*100
print('Training accuracy: ', train_accuracy_SVC, '%')
print('Training time: ', round(time.time() - start_time), ' seconds')
```

```
Training accuracy:  100.0 %
Training time:  388  seconds
```

Problem 3. Evaluation

```
# \*write your codes for evaluation (You can use multiple cells, this is just a place holder)
```

Print the train and test accuracy of your model - Printed Above

```
# Don't hardcode the train and test accuracy
print('Training accuracy for LSTM is %2.3f :' %(train_accuracy) )
```

```
Training accuracy for LSTM is 100.000 :
```

```
print('Test accuracy for LSTM is %2.3f :' %(test_accuracy) )
```

```
Test accuracy for LSTM is 80.967 :
```

Print the train and test and test accuracy of SVM

```
# Don't hardcode the train and test accuracy
print('Training accuracy for SVC is %2.3f' %(train_accuracy_SVC))
```

```
print('Training accuracy for SVC is %2.3f' %(train_accuracy_SVC))
```

```
Training accuracy for SVC is 100.000
```

```
print('Test accuracy for SVC is %2.3f' %(test_accuracy_SVC))
```

```
Test accuracy for SVC is 86.015
```

Problem 4. Report

Bonus

Print the size of your training and test data

```
# Don't hardcode the shape of train and test data
print('Shape of training data is :', )
print('Shape of test/validation data is :', )
```

Modelling and evaluation

```
#Write your code for modelling and evaluation
```

Submission

Runnable source code in ipynb file and a pdf report are required.

The report should be of 3 to 4 pages describing what you have done and learned in this homework and report performance of your model. If you have tried multiple methods, please compare your results. If you are using any external code, please cite it in your report. Note that this homework is designed to help you explore and get familiar with the techniques. The final grading will be largely based on your prediction accuracy and the different methods you tried (different architectures and parameters).

Please indicate clearly in your report what model you have tried, what techniques you applied to improve the performance and report their accuracies. The report should be concise and include the highlights of your efforts. The naming convention for report is **Surname_Givenname_SBUID_report*.pdf**

When submitting your .zip file through blackboard, please -- name your .zip file as **Surname_Givenname_SBUID_hw*.zip**.

This zip file should include:

```
Surname_Givenname_SBUID_hw*
        |---Surname_Givenname_SBUID_hw*.ipynb
        |---Surname_Givenname_SBUID_hw*.pdf
        |---Surname_Givenname_SBUID_report*.pdf
```

For instance, student Michael Jordan should submit a zip file named "Jordan_Michael_111134567_hw5.zip" for homework5 in this structure:

```
Jordan_Michael_111134567_hw5
        |---Jordan_Michael_111134567_hw5.ipynb
        |---Jordan_Michael_111134567_hw5.pdf
        |---Jordan_Michael_111134567_report*.pdf
```

The **Surname_Givenname_SBUID_hw*.pdf** should include a **google shared link**. To generate the **google shared link**, first create a folder named **Surname_Givenname_SBUID_hw*** in your Google Drive with your Stony Brook account.

Then right click this folder, click ***Get shareable link***, in the People textfield, enter two TA's emails: ***bo.cao.1@stonybrook.edu*** and ***sayontan.ghosh@stonybrook.edu***. Make sure that TAs who have the link **can edit**, ***not just* can view**, and also **uncheck** the **Notify people** box.

Colab has a good feature of version control, you should take advantage of this to save your work properly. However, the timestamp of the submission made in blackboard is the only one that we consider for grading. To be more specific, we will only grade the version of your code right before the timestamp of the submission made in blackboard.

You are encouraged to post and answer questions on Piazza. Based on the amount of email that we have received in past years, there might be dealys in replying to personal emails. Please ask questions on Piazza and send emails only for personal issues.

Be aware that your code will undergo plagiarism check both vertically and horizontally. Please do your own work.

Drive Link: https://drive.google.com/drive/folders/1Wi5pmgQ_hBVxGJOBYf943Q5kc01ff6fI?usp=sharing

/