Assignment-5 Computer Vision
Action Recognition

In this assignment, we are loading UCF101 dataset for collecting images from 101 videos,relating to different actions. Each action spans over 110+ clips with each clip provided to us as a folder. Each folder/clip has 25 frames , making it 13320 images(frames) in total.

Below are initial 10 classes provided to us in `annos/actions.txt` file.

1. ApplyEyeMakeup
2. ApplyLipstick
3. Archery
4. BabyCrawling
5. BalanceBeam
6. BandMarching
7. BaseballPitch
8. Basketball
9. BasketballDunk
10. BenchPress

`annos/videos_labels_subsets.txt` helps us separate our image data into training and testing set. On examining this file closely , we figure out that total video folders corresponding to first 25 classes is 3625 amongst 13320 totally available folders.

- lists all the videos (`v_000001`, .., `v_013320`)
- labels (`1`, .., `101`)
- subsets (`1` for train, `2` for test)

Feature Extraction:

Size of the raw images is 256x340, whereas our pre-trained VGG model takes **224x224 sized** 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.
VGG Architecture can be found in the next page.

For first 25 classes:
Each image is a tensor of shape(1,4096).
Total data is a tensor of shape(3360,25,4096) where 25 is the no of images in each folder. 3360 is the #video folders.

For our 25 class data, we generate train:test split of 2409:951 (almost 3:1 ratio, 0.7169642857 to be exact.)

We then use pickle library in Python3.6 to save our created Features (in .mat file) as <pickle_file_name>.p for both our train and test images/labels. (helps in easy loading/unloading files from GDrive)

We then shuffle our training data combined with labels to generate batches for our LSTM model to train on. The results of training different architectures of our LSTM models have been analysed towards the end of this report.

VGG Modelling:

```
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)
  )
```

We use this VGG model over each 5 crops of all the images and push the resultant tensor into a feature vector. We normalize each of our images using Pytorch torchvision's library transform(), giving it the mean and std_dev as follows:

```
mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
```

LSTM Models:

```
Model1(
  (recurrent_layer1): LSTM(4096, 500, batch_first=True, dropout=0.4)
  (classify_layer1): Linear(in_features=500, out_features=25, bias=True)
)

Model2(
  (recurrent_layer1): LSTM(4096, 500, batch_first=True, dropout=0.3)
  (d): Dropout(p=0.4, inplace=False)
  (classify_layer1): Linear(in_features=500, out_features=25, bias=True)
)

Model3(
  (recurrent_layer1): LSTM(4096, 500, batch_first=True, dropout=0.4)
  (act): ReLU(inplace=True)
  (classify_layer1): Linear(in_features=500, out_features=25, bias=True)
)


Model4(
  (recurrent_layer): LSTM(4096, 100, batch_first=True, dropout=0.5)
  (classify_layer): Sequential(
    (0): Conv1d(25, 4, kernel_size=(3,), stride=(1,))
    (1): BatchNorm1d(4, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Flatten()
    (5): Linear(in_features=392, out_features=25, bias=True)
  )
)
```

	#Hidden Layers	Dropout	Optimizer	Learning rate	#Epochs	Final Loss	Accuracy
Model 1	1	0.4	Adam	1e-3	20	0.1579	79.22
Model 2	1	0.3+0.4	Adam	1e-3	20	0.4046	77.95
Model 3	**1**	**0.4**	**Adam**	**1e-4**	**20**	**0.0009**	**85.86**
Model 4	1	0.5	Adam	1e-3	20	0.1167	79.96

Table:Comparison between LSTM Models trained using different hyperparameters

This proves our **Model 3** to be the **best** performing for the given UCF101 dataset, when trained on 25 action class sequences of image data(videos). We have achieved an accuracy of almost 86% with final loss of 0.0009 , a significant improvement over the other models.
This accuracy boost can be attributed to the use of Non-linearity (***Rectified Linear Unit/ ReLu***) applied on top of our recurrent layer after dropout.

We have used **Adam optimizer** , over SGD due to following reasons:

- In the case of Adam ,the weights are updated once at the end of each epoch, whereas in SGD, the weights are updated after looping via each training sample, thereby requiring more fine tuning at each step.

- Adam adaptively select a separate learning rate for each parameter. Parameters that would ordinarily receive smaller or less frequent updates receive larger updates with Adam (the reverse is also true). This speeds learning in cases where the appropriate learning rates vary across parameters.

- Another benefit to this approach is that, because learning rates are adjusted automatically, manual tuning becomes less important. Standard SGD requires careful tuning (and possibly online adjustment) of learning rates, but this less true with Adam and related methods. It's still necessary to select hyperparameters, but performance is less sensitive to them than to SGD learning rates

Comparison with SVM:

We train our SVM 1 vs All classifier on 25 class video frames , by flattening each 2d image into 1d, and stacking vertically in a numpy array.

SVM Model Architecture:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

On comparing with SVM classifier, over same 25 class images, we find out the SVM results in a Accuracy of **82.75499474237644 %** over the test set, which is comparable to our trained LSTM's accuracy(Leading over 3 Models, trailing behind just 1 (Model3).

Bonus: (Training on 101 class videos)
Picking best model trained on 25 classes(small dataset).
LSTM Model 3: Ran on entire training data set of 101 class videos. Achieved accuracy of 76.67% on Test set.

```
Epoch: 16  , Loss: 0.0029,  Accuracy: 100.00
Test set accuracy is
Got 2898 / 3780 correct (76.67)

----x-----x----- x-----x-----x-----x-----x-----x-----x-----x-----x-----x---
```

References:

1. https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html#sphx-glr-beginner-nlp-sequence-models-tutorial-py
2. https://blog.floydhub.com/a-beginners-guide-on-recurrent-neural-networks-with-pytorch/