# 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 (https://pytorch.org/docs/stable/torchvision/models.html)](https://pytorch.org/docs/stable/torchvision/models.html) 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, 0.2
    24, 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 (http://pytorch.org/tutorials/beginner/data_loading_tutorial.html)](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 (http://vision.cs.stonybrook.edu/~yangwang/public/UCF101_images.tar)](http://vision.cs.stonybrook.edu/~yangwang/public/UCF101_images.tar)(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://blog.echen.me)](http://blog.echen.me)
  - [http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (http://karpathy.github.io/2015/05/21/rnn-effectiveness/)](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)
  - [http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)
- Implementing RNN and LSTM with PyTorch
  - [LSTM with PyTorch (http://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html#sphx-glr-beginner-nlp-sequence-models-tutorial-py)](http://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html#sphx-glr-beginner-nlp-sequence-models-tutorial-py)
  - [RNN with PyTorch (http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html)](http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html)

In [0]:
```
# write your codes here
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import transforms, models
import numpy as np
import scipy as sp
import pandas as pd
from cv2 import imread
from scipy.io import savemat, loadmat
import os, glob, time, random, cv2
```

In [3]:
```
# Mount your google drive where you've saved your assignment folder
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

In [0]:
```
# import tarfile
# fname = 'images/UCF101_images.tar'
# if (fname.endswith("tar.gz")):
#     tar = tarfile.open(fname, "r:gz")
#     tar.extractall()
#     tar.close()
# elif (fname.endswith("tar")):
#     tar = tarfile.open(fname, "r:")
#     tar.extractall()
#     tar.close()
```

In [0]:
```
# fname = 'temp.zip'

# import zipfile
# with zipfile.ZipFile(fname, 'r') as zip_ref:
#     zip_ref.extractall()
```

In [0]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

In [0]:
```
info = pd.read_csv('annos/videos_labels_subsets.txt', header=None, delimiter='
\t')
```

In [0]:
```
def tile_image(img, tile_size=(224, 224)):
    return (img[:, :tile_size[0], :tile_size[1]], img[:, (img.shape[1]-tile_si
ze[0]):, :tile_size[1]],
            img[:, :tile_size[0], (img.shape[2]-tile_size[1]):],
            img[:, (img.shape[1]-tile_size[0]):, (img.shape[2]-tile_size[1
]):],
            img[:, (img.shape[1]-tile_size[0])//2:(img.shape[1]+tile_size[0])
//2, (img.shape[2]-tile_size[1])//2:(img.shape[2]+tile_size[1])//2])
```

# Problem 1. Feature extraction

In [0]:
```
# \*write your codes for feature extraction (You can use multiple cells, this
 is just a place holder)
```

In [5]:
```
vgg16 = models.vgg16(True).to(device)
del vgg16.classifier[2:]
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224
, 0.225])
prep = transforms.Compose([ transforms.ToTensor(), normalize ])
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /roo
t/.cache/torch/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:08<00:00, 61.9MB/s]
```

In [0]:
```
nclasses = 25
classes = info[1].unique()[:nclasses]

train_folders_names = []
test_folders_names = []
train_labels = []
test_labels = []

for classindex in classes:
    train_folders = info[(info[1] == classindex) & (info[2] == 1)][0].tolist()
    train_folders_names.extend(train_folders)

    test_folders = info[(info[1] == classindex) & (info[2] == 2)][0].tolist()
    test_folders_names.extend(test_folders)

    train_labels.extend(np.repeat(classindex, len(train_folders)))
    test_labels.extend(np.repeat(classindex, len(test_folders)))
```

In [0]:
```
def extract_save(folders, mode, source = 'images', target = 'temp'):
    flag = 0
    total = len(folders)
    train_data = np.array([])

    start = time.time()
    for i, folder in enumerate(folders):

        if os.path.exists(os.path.join(target, mode, folder+'.mat')):
            # print('skipping', folder+'.mat')
            continue
        flag = 1
        features = []
        for f in os.listdir(os.path.join(source, folder)):

            fname = os.path.join(source, folder, f)
            print(imread(fname).min())
            img = prep(imread(fname)).to(device)
            print(img.min())
            with torch.no_grad():
                print(torch.stack(tile_image(img)).shape)
                import sys
                sys.exit()
                img = vgg16(torch.stack(tile_image(img)))
                features.append(img.mean(dim=0).cpu()[...,np.newaxis])

        features = np.concatenate(features, axis=-1)
        # savemat(os.path.join(target, mode, folder+'.mat'), {'Feature':featur
es})

    if flag != 1:
        print("Nothing new!")
    print('total time', time.time() - start)
```

In [10]:
```
extract_save(source = 'images', target = 'temp', folders = train_folders_names
, mode = 'train')
```

```
Nothing new!
total time 1.913640022277832
```

In [0]:
```
extract_save(source = 'images', target = 'temp', folders = test_folders_names,
mode = 'test')
```

```
Nothing new!
total time 1.7258226871490479
```

In [0]:
```
target = 'temp'
def load_data(folders, mode):
    data = []
    for i, folder in enumerate(folders):
        if i%1000==0:
          print(i)
        features = loadmat(os.path.join(target, mode, folder+'.mat'))['Featur
e']
        data.append(features)
    return np.array(data)
```

In [0]:
```
train_data = load_data(train_folders_names, 'train')
```

In [0]:
```
test_data = load_data(test_folders_names, 'test')
```

---

---

# Problem 2. Modelling

- ##### Print the size of your training and test data

In [11]:
```
# Don't hardcode the shape of train and test data
print('Shape of training data is :', np.array(train_data).shape)
print('Shape of test/validation data is :', np.array(test_data).shape)
```

```
Shape of training data is : (2409, 4096, 25)
Shape of test/validation data is : (951, 4096, 25)
```

In [0]:
```
def evaluate(model, data, labels):
    with torch.no_grad():
        correct = 0
        total = 0
        for video, labels in zip(data, labels):
            video, labels = torch.tensor(video).to(device), labels
            outputs = model(video).cpu()
            predicted = np.argmax(outputs.numpy())
            total += 1

            correct += (predicted == labels-1).sum().item()
    return 100 * correct / total
```

In [0]:
```
# \*write your codes for modelling using the extracted feature (You can use mu
ltiple cells, this is just a place holder)

def train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data, train
_labels, test_data, test_labels):
    num_subclass = nclasses
    class LSTMAction(nn.Module):
        def __init__(self, feature_dim, hidden_dim, action_size, drop_prob=dro
p_prob):
            super(LSTMAction, self).__init__()
            self.hidden_dim = hidden_dim
            self.lstm = nn.LSTM(feature_dim, hidden_dim, n_layers) #, dropout
 = drop_prob
            self.final = nn.Linear(hidden_dim, action_size)
            self.hidden = self.init_hidden()

        def init_hidden(self):
            return (torch.zeros(n_layers, 1, self.hidden_dim).to(device),
                        torch.zeros(n_layers, 1, self.hidden_dim).to(device))

        def forward(self, video):
            lstm_out, self.hidden = self.lstm(video.view(25, 1, -1), self.hidd
en)
            output = self.final(self.hidden[-1])
            return output
        # def __init__(self, feature_dim, num_hidden, nclasses, drop_prob=drop
_prob):
        #     super(LSTMAction, self).__init__()
        #     self.num_hidden = num_hidden
        #     self.lstm = nn.LSTM(feature_dim, num_hidden, n_layers, dropout =
drop_prob) #
        #     self.dense = nn.Linear(num_hidden, 512)
        #     self.final = nn.Linear(512, nclasses)
        #     self.hidden = self.init_hidden()

        # def init_hidden(self):
        #     return (torch.zeros(n_layers, 1, self.num_hidden).to(device),
        #                 torch.zeros(n_layers, 1, self.num_hidden).to(devic
e))

        # def forward(self, video):
        #     lstm_out, self.hidden = self.lstm(video.view(25, 1, -1), self.hi
dden)
        #     output = self.final(self.dense(F.relu(self.hidden[0])))
        #     return output

    train = ()
    model = LSTMAction(4096, num_hidden, num_subclass)

    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.09)
    model.to(device)
    model(torch.tensor(train_data[0]).to(device))

    print("Training started!")
    for epoch in range(epochs):
```

```
        c = list(zip(train_data, train_labels))
        random.shuffle(c)
        train_data, train_labels = zip(*c)

        for data, label in zip(train_data, train_labels):
            model.zero_grad()
            model.hidden = model.init_hidden()

            label = torch.tensor(label-1)
            data, label = torch.tensor(data).to(device), label.to(device)
            tag_scores = model(data)
            loss = loss_function(tag_scores.view(1, n_layers * num_subclass),
label.view(1))
            loss.backward()
            optimizer.step()
        train_acc = evaluate(model, train_data, train_labels)
        val_acc   = evaluate(model, test_data, test_labels)
        print('Epoch %d => train acc: %d, val acc: %d %%' % (epoch+1, train_ac
c, val_acc))
    print("Training completed!")
    return model
```

In [0]:

```
num_hidden = 64
n_layers = 3
epochs = 10
drop_prob = 0.5
lr = 0.01
model25 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data,
train_labels, test_data, test_labels)
```

```
Training started!
Epoch 1 => train acc: 79, val acc: 63 %
Epoch 2 => train acc: 94, val acc: 72 %
Epoch 3 => train acc: 98, val acc: 75 %
Epoch 4 => train acc: 99, val acc: 75 %
Epoch 5 => train acc: 100, val acc: 76 %
Epoch 6 => train acc: 100, val acc: 76 %
Epoch 7 => train acc: 100, val acc: 77 %
Epoch 8 => train acc: 100, val acc: 78 %
Epoch 9 => train acc: 100, val acc: 78 %
Epoch 10 => train acc: 100, val acc: 78 %
Training completed!
```

In [25]:
```
num_hidden = 512
n_layers = 2
epochs = 10
drop_prob = 0.5
lr = 0.01
model25 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data,
train_labels, test_data, test_labels)
```

```
Training started!
Epoch 1 => train acc: 94, val acc: 73 %
Epoch 2 => train acc: 98, val acc: 75 %
Epoch 3 => train acc: 100, val acc: 78 %
Epoch 4 => train acc: 100, val acc: 78 %
Epoch 5 => train acc: 100, val acc: 77 %
Epoch 6 => train acc: 100, val acc: 78 %
Epoch 7 => train acc: 100, val acc: 78 %
Epoch 8 => train acc: 100, val acc: 78 %
Epoch 9 => train acc: 100, val acc: 78 %
Epoch 10 => train acc: 100, val acc: 78 %
Training completed!
```

In [53]:
```
num_hidden = 1012
n_layers = 2
epochs = 25
drop_prob = 0.3
lr = 0.001
model25 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data,
train_labels, test_data, test_labels)
```

```
Training started!
Epoch 1 => train acc: 27, val acc: 23 %
Epoch 2 => train acc: 43, val acc: 37 %
Epoch 3 => train acc: 68, val acc: 57 %
Epoch 4 => train acc: 82, val acc: 72 %
Epoch 5 => train acc: 88, val acc: 74 %
Epoch 6 => train acc: 94, val acc: 75 %
Epoch 7 => train acc: 96, val acc: 76 %
Epoch 8 => train acc: 98, val acc: 76 %
Epoch 9 => train acc: 98, val acc: 76 %
Epoch 10 => train acc: 99, val acc: 77 %
Epoch 11 => train acc: 100, val acc: 75 %
Epoch 12 => train acc: 100, val acc: 77 %
Epoch 13 => train acc: 100, val acc: 77 %
Epoch 14 => train acc: 100, val acc: 76 %
Epoch 15 => train acc: 100, val acc: 77 %
Epoch 16 => train acc: 100, val acc: 76 %
Epoch 17 => train acc: 100, val acc: 76 %
Epoch 18 => train acc: 100, val acc: 77 %
Epoch 19 => train acc: 100, val acc: 77 %
Epoch 20 => train acc: 100, val acc: 76 %
Epoch 21 => train acc: 100, val acc: 76 %
Epoch 22 => train acc: 100, val acc: 78 %
Epoch 23 => train acc: 100, val acc: 76 %
Epoch 24 => train acc: 100, val acc: 77 %
Epoch 25 => train acc: 100, val acc: 76 %
Training completed!
```

In [39]:
```
num_hidden = 1024
n_layers = 5
epochs = 10
drop_prob = 0.3
lr = 0.001
model25 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data,
train_labels, test_data, test_labels)
```

```
Training started!
Epoch 1 => train acc: 83, val acc: 70 %
Epoch 2 => train acc: 93, val acc: 75 %
Epoch 3 => train acc: 96, val acc: 74 %
Epoch 4 => train acc: 99, val acc: 80 %
Epoch 5 => train acc: 99, val acc: 79 %
Epoch 6 => train acc: 99, val acc: 79 %
Epoch 7 => train acc: 100, val acc: 80 %
Epoch 8 => train acc: 100, val acc: 80 %
Epoch 9 => train acc: 100, val acc: 80 %
Epoch 10 => train acc: 100, val acc: 80 %
Training completed!
```

# Problem 3. Evaluation

In [40]:
```
# \*write your codes for evaluation (You can use multiple cells, this is just
 a place holder)

train_acc = evaluate(model25, train_data, train_labels)
test_acc   = evaluate(model25, test_data, test_labels)
print('Accuracy of the network on the train data: %d %%' % (train_acc))
print('Accuracy of the network on the test data: %d %%' % (test_acc))
```

```
Accuracy of the network on the train data: 99 %
Accuracy of the network on the test data: 80 %
```

In [0]:
```
from sklearn.svm import LinearSVC
svcmodel = LinearSVC()

svctrain = []
svclabel = []
for data, label in zip(train_data, train_labels):
    data = torch.tensor(data)
    label = torch.tensor(label)
    svctrain.append(data.view(1, -1))
    svclabel.append(label.view(1, -1))
```

In [0]:
```
svctrain = torch.cat(svctrain).numpy()
svclabel = torch.cat(svclabel).view(-1).numpy()
```

In [0]:
```
svcmodel.fit(svctrain, svclabel)
```

Out[0]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

In [0]:
```
svctrain_results = svcmodel.predict(svctrain)
svcTrainAcc = 100.0*np.sum(svctrain_results == svclabel) / len(svctrain_result
s)
print("Training Accuracy for SVC model: %f %%" % (svcTrainAcc))
```

```
Training Accuracy for SVC model: 100.000000 %
```

In [0]:
```
svctest = []
svctestlabel = []
for data, label in zip(test_data, test_labels):
    data = torch.tensor(data)
    label = torch.tensor(label)
    svctest.append(data.view(1, -1))
    svctestlabel.append(label)
```

In [0]:
```
svctest = torch.cat(svctest).numpy()
svctestlabel = np.array(svctestlabel)
```

In [0]:
```
svctest_results = svcmodel.predict(svctest)
svcAcc = 100.0*np.sum(svctest_results == svctestlabel) / len(svctest_results)
print("Test Accuracy for SVC model: %f %%" % (svcAcc))
```

```
Test Accuracy for SVC model: 86.014721 %
```

- ##### **Print the train and test accuracy of your model**

In [58]:
```
# Don't hardcode the train and test accuracy
print('Training accuracy is %2.3f' %(train_acc) )
print('Test accuracy is %2.3f' %(test_acc) )
```

```
Training accuracy is 99.958
Test accuracy is 80.336
```

- ##### **Print the train and test and test accuracy of SVM**

In [0]:
```
# Don't hardcode the train and test accuracy
print('Training accuracy is %2.3f' %(svcTrainAcc) )
print('Test accuracy is %2.3f' %(svcAcc) )
```

```
Training accuracy is 100.000
Test accuracy is 86.015
```

# Problem 4. Report

# Bonus

In [0]:
```
nclasses = 101
classes101 = info[1].unique()[:nclasses]

train_folders_names101 = []
test_folders_names101 = []
train_labels101 = []
test_labels101 = []

for classindex in classes101:
    train_folders = info[(info[1] == classindex) & (info[2] == 1)][0].tolist()
    train_folders_names101.extend(train_folders)

    test_folders = info[(info[1] == classindex) & (info[2] == 2)][0].tolist()
    test_folders_names101.extend(test_folders)

    train_labels101.extend(np.repeat(classindex, len(train_folders)))
    test_labels101.extend(np.repeat(classindex, len(test_folders)))
```

In [13]:
```
train_data101 = load_data(train_folders_names101, 'train') #9537
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
```

In [14]:
```
test_data101 = load_data(test_folders_names101, 'test') #3783
```

```
0
1000
2000
3000
```

- **##### Print the size of your training and test data**

In [12]:
```
# Don't hardcode the shape of train and test data
print('Shape of training data is :', train_data101.shape)
print('Shape of test/validation data is :', test_data101.shape)
```

```
Shape of training data is : (9537, 4096, 25)
Shape of test/validation data is : (3783, 4096, 25)
```

- ##### **Modelling and evaluation**

In [0]:
```
# \*write your codes for modelling using the extracted feature (You can use mu
ltiple cells, this is just a place holder)

def train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data, train
_labels, test_data, test_labels):
    nclasses=101
    class LSTMAction(nn.Module):
        # def __init__(self, feature_dim, hidden_dim, action_size, drop_prob=d
rop_prob):
        #     super(LSTMAction, self).__init__()
        #     self.hidden_dim = hidden_dim
        #     self.lstm = nn.LSTM(feature_dim, hidden_dim, n_layers)
        #     self.final = nn.Linear(hidden_dim, action_size)
        #     self.hidden = self.init_hidden()

        # def init_hidden(self):
        #     return (torch.zeros(n_layers, 1, self.hidden_dim).to(device),
        #                 torch.zeros(n_layers, 1, self.hidden_dim).to(devic
e))

        # def forward(self, video):
        #     lstm_out, self.hidden = self.lstm(video.view(25, 1, -1), self.hi
dden)
        #     output = self.final(self.hidden[-1])
        #     return output

        def __init__(self, feature_dim, num_hidden, nclasses, drop_prob=drop_p
rob):
            super(LSTMAction, self).__init__()
            self.num_hidden = num_hidden
            self.lstm = nn.LSTM(feature_dim, num_hidden, n_layers, dropout = d
rop_prob)
            self.dense = nn.Linear(num_hidden, 512)
            self.final = nn.Linear(512, nclasses)
            self.hidden = self.init_hidden()

        def init_hidden(self):
            return (torch.zeros(n_layers, 1, self.num_hidden).to(device),
                        torch.zeros(n_layers, 1, self.num_hidden).to(device))

        def forward(self, video):
            lstm_out, self.hidden = self.lstm(video.view(25, 1, -1), self.hidd
en)
            output = self.final(self.dense(F.relu(self.hidden[0])))
            return output

    train = ()
    model = LSTMAction(4096, num_hidden, nclasses)

    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.09)
    model.to(device)
    model(torch.tensor(train_data[0]).to(device))

    print("Training started!")
    for epoch in range(epochs):
```

```
        c = list(zip(train_data, train_labels))
        random.shuffle(c)
        train_data, train_labels = zip(*c)

        for data, label in zip(train_data, train_labels):
            model.zero_grad()
            model.hidden = model.init_hidden()

            label = torch.tensor(label-1)
            data, label = torch.tensor(data).to(device), label.to(device)
            tag_scores = model(data)
            loss = loss_function(tag_scores.view(1, n_layers * nclasses), labe
l.view(1))
            loss.backward()
            optimizer.step()
        train_acc = evaluate(model, train_data, train_labels)
        val_acc   = evaluate(model, test_data, test_labels)
        print('Epoch %d => train acc: %d, val acc: %d %%' % (epoch+1, train_ac
c, val_acc))
    print("Training completed!")
    return model
```

In [22]:

```
#Write your code for modelling and evaluation

num_hidden = 64
n_layers = 3
epochs = 10
lr = 0.01
model101 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data
101, train_labels101, test_data101, test_labels101)
```

```
Training started!
Epoch 1 => train acc: 73, val acc: 49 %
Epoch 2 => train acc: 85, val acc: 52 %
Epoch 3 => train acc: 92, val acc: 52 %
Epoch 4 => train acc: 58, val acc: 26 %
Epoch 5 => train acc: 95, val acc: 46 %
Epoch 6 => train acc: 98, val acc: 54 %
Epoch 7 => train acc: 98, val acc: 54 %
Epoch 8 => train acc: 98, val acc: 54 %
Epoch 9 => train acc: 98, val acc: 53 %
Epoch 10 => train acc: 98, val acc: 52 %
Training completed!
```

In [19]:
```
#Write your code for modelling and evaluation

num_hidden = 256
n_layers = 2
epochs = 10
drop_prob = 0.5
lr = 0.01
model101 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data
101, train_labels101, test_data101, test_labels101)
```

```
Training started!
Epoch 1 => train acc: 64, val acc: 46 %
Epoch 2 => train acc: 80, val acc: 50 %
Epoch 3 => train acc: 89, val acc: 55 %
Epoch 4 => train acc: 95, val acc: 57 %
Epoch 5 => train acc: 95, val acc: 55 %
Epoch 6 => train acc: 99, val acc: 60 %
Epoch 7 => train acc: 100, val acc: 61 %
Epoch 8 => train acc: 100, val acc: 61 %
Epoch 9 => train acc: 100, val acc: 61 %
Epoch 10 => train acc: 100, val acc: 61 %
Training completed!
```

In [19]:
```
#Write your code for modelling and evaluation

num_hidden = 512
n_layers = 3
epochs = 10
drop_prob = 0.5
lr = 0.01
model101 = train_model(num_hidden, n_layers, epochs, drop_prob, lr, train_data
101, train_labels101, test_data101, test_labels101)
```

```
Training started!
Epoch 1 => train acc: 60, val acc: 44 %
Epoch 2 => train acc: 79, val acc: 53 %
Epoch 3 => train acc: 87, val acc: 54 %
Epoch 4 => train acc: 94, val acc: 56 %
Epoch 5 => train acc: 99, val acc: 61 %
Epoch 6 => train acc: 99, val acc: 62 %
Epoch 7 => train acc: 100, val acc: 64 %
Epoch 8 => train acc: 100, val acc: 64 %
Epoch 9 => train acc: 100, val acc: 64 %
Epoch 10 => train acc: 100, val acc: 64 %
Training completed!
```

In [20]:
```
train_acc101 = evaluate(model101, train_data101, train_labels101)
test_acc101   = evaluate(model101, test_data101, test_labels101)
print('Accuracy of the network on the train data: %d %%' % (train_acc101))
print('Accuracy of the network on the test data: %d %%' % (test_acc101))
```

```
Accuracy of the network on the train data: 100 %
Accuracy of the network on the test data: 64 %
```