Introduction to Deep Learning

Name: **Due Date:** May. 12, 2021

Final Project

Problem Description

Recognizing human action from video data can be quite challenging because of variations in camera motion, unexpected object appearance and poses, viewpoints, cluttered background and illumination conditions, etc. The University of Central Florida (UCF) YouTube action data set contains 11 action categories for basketball shooting, biking,/cycling, diving, golf swinging, horse riding, soccer juggling, swinging, tennis swinging, trampoline jumping, volleyball spiking and walking with a dog. These videos can be processes into image sequences and used to train a machine learning model to predict the human action of each video.

Model Description

A model containing convolutional layers, recurrent layers (Long-short-term memory LSTM) and a fully connected dense layer is used. The structure of the model is as follows:

- An input layer with shape [Batch-size, 30, 64, 64, 3]
- A Convolution layer with filter size [5, 5, 32] and an output shape of [Batch-size, 30, 60, 60, 32] and a ReLU activation function.
- A 3D Max pooling layer with pool size of $(1, 2, 2)$ and stride $[1, 2, 2]$. The output shape is [Batch-size, 30, 30, 30, 32].
- A Convolution layer with filter size [3, 3, 64] and an output shape of [Batch-size, 30, 28, 28, 64] and a ReLU activation function.
- A 3D Max pooling layer with pool size of $(1, 2, 2)$ and stride $[1, 2, 2]$. The output shape is [Batch-size, 30, 14, 14, 64].
- A reshape layer to flatten the the last 3 dimensions of the tensor to shape $[batch_size, 30, -1]$.
- An LSTM layer with 100 units and return sequence set to true. Default activation functions of standard tensorflow LSTM module was used for the gates.
- An LSTM layer with 50 units and return sequence set to False. Default activation functions of standard tensorflow LSTM module was used for the gates.

- A fully connected layer of 11 nodes. With a softmax activation function.

A forward pass of this model is implemented by feeding the image data into the network through the input layer. The convolution layers were implemented using tf.keras.layers.Conv2D() and the filter size, padding and activation arguments are passed along.
The pooling layers were implemented using tf.keras.layers.MaxPool3D(), the reshape layer was implemented using tf.keras.layers.Reshape(), the LSTM layer was implemented using tf.keras.layers.LSTM, while the fully connected layer was implemented using tf.keras.layers.Dense().
The **loss function** used is the categorical cross-entropy. Implemented in tensorflow by calling tf.keras.losses.categorical_crossentropy.

Experimental setting

- A laptop with 16gb CPU and Nvidia RTX 2070 (8gb) was used.
- Python 3.8 was used for coding.
- Training and testing data were imported and processed using the hints provided. The data were normalized by dividing with 255.0. The training data was split into a validation set and 4 sub training data since the local machine could not hold the entire training data in it memory and still train.
- Tensorflow version 2.4.0 was to build the model function using tf.keras.models.Sequential
- the model architecture was set during call of function Model with the input dimension
- The gradients of the weights were computed using tensorflow's tf.GradientTape() function. The loss function used is the mean squared error; tf.keras.losses.categorical_crossentropy.
- Adam optimizer with learning rate $\eta = 0.0001$ was initially used for training. The optimizer was also used to apply the gradients to the model's trainable variables.
- Model is initialized by calling the Model function.
- Each of the sub-training data is shuffled and batches are generated using tf.data.Dataset.from_tensor_slices function.
- Although training loss was calculated during each weight update, the training and testing losses and accuracy were recorded after each epoch.

Hyper-parameters Used

- batch size = 10, No of batches = $1500/10 \times 4 = 600$
- Epoch = 30

- Number of iterations: No of batches $\times$ Epoch $= 600 \times 30 = 18000$
- learning rate $\eta = 0.0001$

Algorithm 1 Training Pseudo-code

Input: train_data, train_label, val_data,val_label, model input_size, learning rate, etc
Output: Trained model, training and validation losses and accuracies

- **Initialize model** by calling the Model function
- **Set** hyper-parameters and optimizer and loss function
- **For** i = 0,1,. . . ,epoch

    load data and **generate** batches by calling the genBatch function

        for X,Y in each training batch,

            Perform a forward propagation using the X

            Compute the loss between prediction and Y

            Compute gradient of the loss for all the data in the batch

            Apply gradients to model trainable variables using the optimizer.

        Save the mean training loss for each batch

    Save the mean training loss for each epoch

    Compute the training accuracy for each epoch

    Using the validation data, perform forward propagation and compute the validation loss and save.

    Compute validation accuracy for each epoch

Training and Validation losses

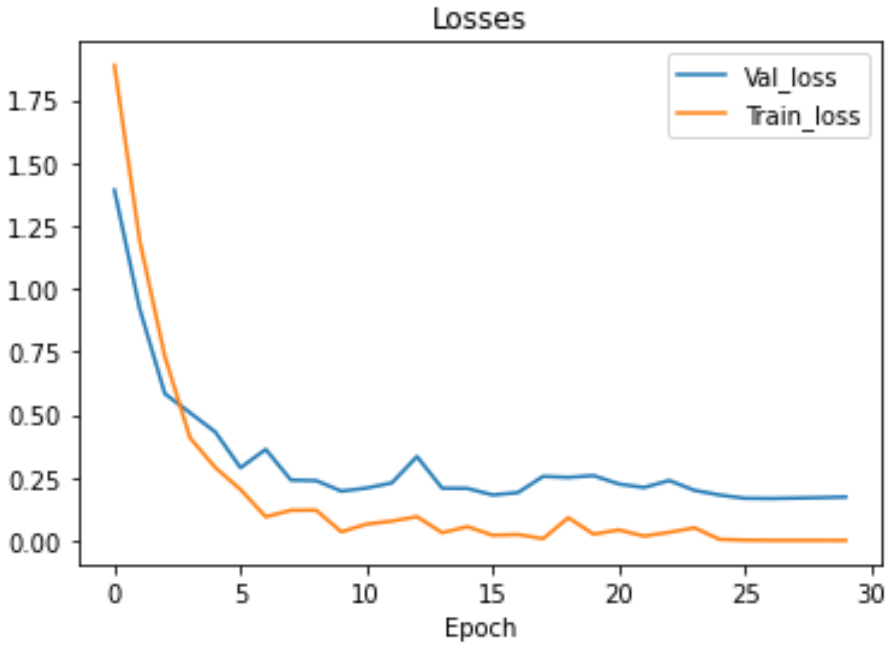

Figure 1: Training and Validation losses

Training and Validation Accuracy

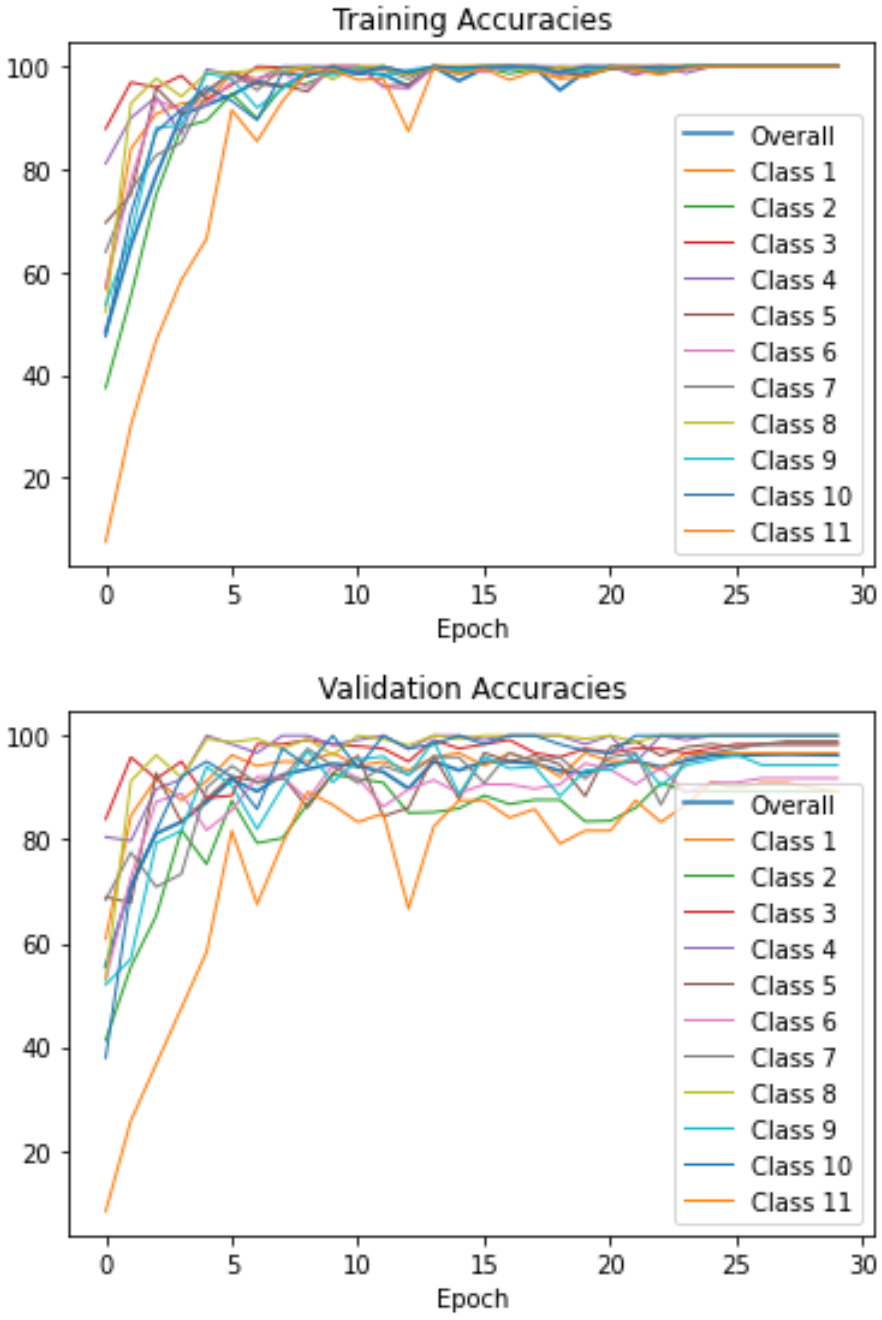

Figure 2: Accuracy Curves

Confusion Matrix

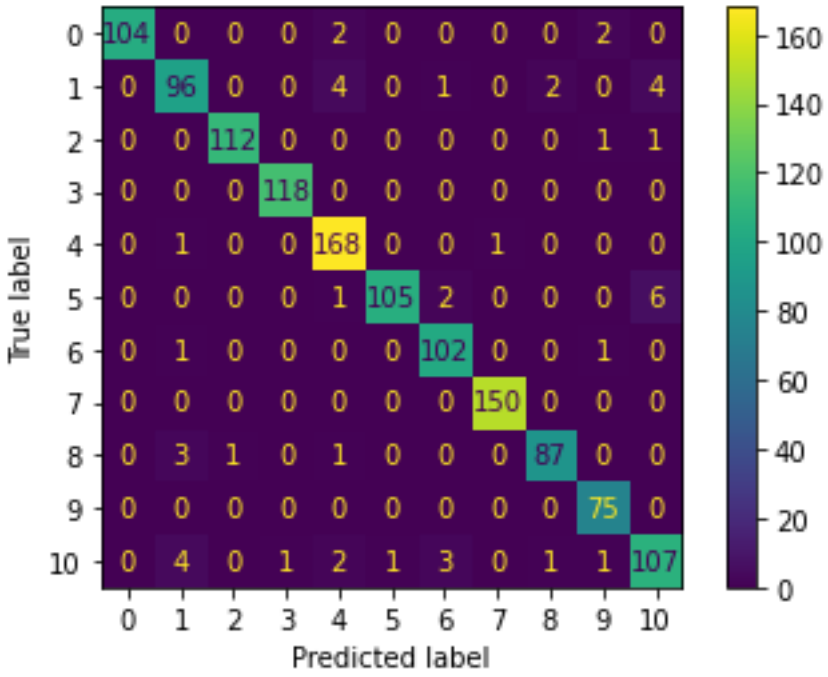

Figure 3: Confusion Matrix

Final Validation Accuracies

Final Accuracies	
Class	Accuracy (%)
1	96.67
2	89.29
3	98.33
4	100
5	98.89
6	91.79
7	98.33
8	100
9	94.33
10	100
11	89.1667
Overall	96.07

Discussion

Initially, a single convolutional layer with 32 filters was used and training losses did not reduce beyond 0.2. Then additional convolutional layer with 64 filters was added to improved on the result. Also, the learning rate was tuned to improve on model performance. Final learning rate used is 0.001.