Deep Learning PA4 Report

Sahith Bhamidipati

March 2022

1 Problem Statement

The purpose of this assignment is to build a deep dynamic model that determines the poses of people in videos. The model is expected to map out the joint positions of a person over an 8-frame video, where the pose in each frame is predicted individually. However, the overall error of a prediction is measured as the average of the error of the pose prediction of each frame based on the euclidean distance between the predicted pose and the true pose in millimeters. The final model that provides a solution to this problem is expected to predict poses of people in videos with an overall error of less than 150 millimeters from the true poses.

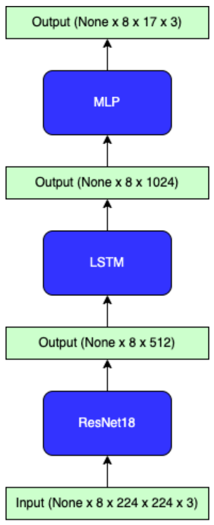
2 Model Description

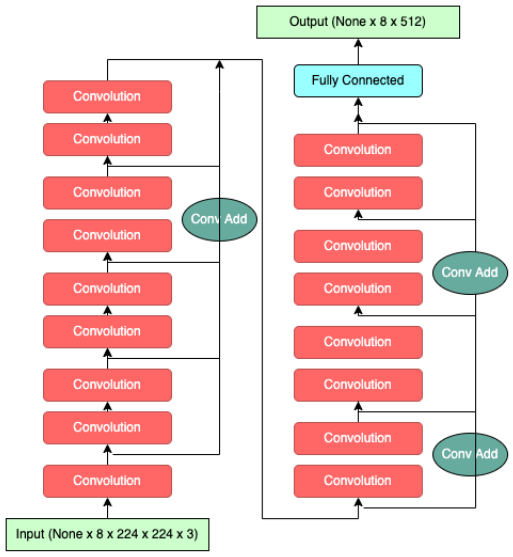
The model used was a deep dynamic model consisting of a ResNet18 model followed by a long short-term memory (LSTM) layer and multiple fully connected layers leading to the output.

First, the ResNet18, which utilizes 18 layers in its architecture, is passed in the original data of shape $N \times 8 \times 224 \times 224 \times 3$, where N is the batch size. This input goes through a convolution layer, then is normalized using batch normalized and passed through the ReLU activation function. Next, after passing through a max pooling layer, the data runs through the remaining convolution layers in two types of blocks: an identity block and a convolution block. Both types of blocks contain two convolution layers each followed by batch normalization and ReLU activation, but the identity block combines its final output with the original input, while the convolution block combines its final output with the original input passed through its own convolution layer. With 8 of these blocks followed by a fully connected layer providing the output, the total count of layers in the model reaches 18, hence ResNet18.

After the ResNet18 outputs an $N \times 8 \times 512$ output, the LSTM layer takes it as input and outputs an $N \times 8 \times 1024$ output. From here, a fully connected layer of 500 nodes and another with 200 nodes, both with ReLU activation, are applied, and the final output is formed after passing through a final fully connected layer of 51 nodes.

The below images show the outer architecture of the entire model (left) and a more detailed view of the ResNet18 architecture (right).





3 Loss Function

The loss function used to evaluate the performance of this model was the mean squared error function. The mean squared error function is given as

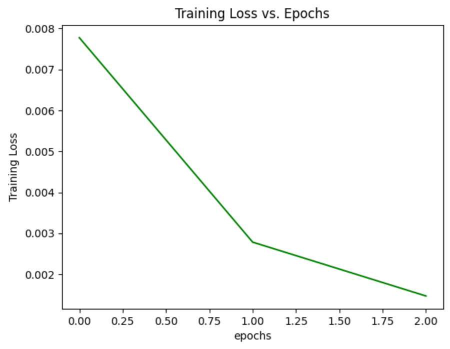
$$MSE = \frac{1}{N}\sum_{n=1}^{N} \|y_n - \hat{y}_n\|^2 = \frac{1}{N*K*D}\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{d=1}^{D}(y_{n,k,d} - \hat{y}_{n,k,d})^2$$

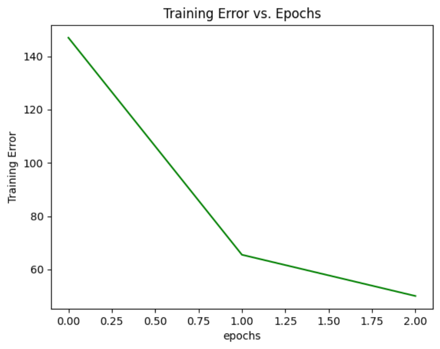
where N is the number of frames in a video (8), K is the number of joints (17), and D is the dimension of the coordinates (3), making y and $\hat{y}$ $8 \times 17 \times 3$ tensors representing the joint position of each joint in each frame of the video. y represents the true joint positions, while $\hat{y}_n$ represents the respective joint position predictions from the model output. The derivative of the loss function, used in back propagation, is

$$\begin{aligned}
\frac{\partial L(y,\hat{y})}{\partial \hat{y}} &= \frac{1}{N*K*D}\frac{\partial \sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{d=1}^{D}(y_{n,k,d} - \hat{y}_{n,k,d})^2}{\partial \hat{y}} \\
&= \frac{1}{N*K*D}\left[\begin{bmatrix} \frac{\partial(y_{1,1,1}-\hat{y}_{1,1,1})^2}{\partial \hat{y}_{1,1,1}} & \dots & \frac{\partial(y_{1,1,3}-\hat{y}_{1,1,3})^2}{\partial \hat{y}_{1,1,3}} \\ \vdots & & \vdots \\ \frac{\partial(y_{1,17,1}-\hat{y}_{1,17,1})^2}{\partial \hat{y}_{1,17,1}} & \dots & \frac{\partial(y_{1,17,3}-\hat{y}_{1,17,3})^2}{\partial \hat{y}_{1,17,3}} \end{bmatrix};\right. \\
&\qquad \left.\left[\dots; \begin{bmatrix} \frac{\partial(y_{8,1,1}-\hat{y}_{8,1,1})^2}{\partial \hat{y}_{8,1,1}} & \dots & \frac{\partial(y_{8,1,3}-\hat{y}_{8,1,3})^2}{\partial \hat{y}_{8,1,3}} \\ \vdots & & \vdots \\ \frac{\partial(y_{8,17,1}-\hat{y}_{8,17,1})^2}{\partial \hat{y}_{8,17,1}} & \dots & \frac{\partial(y_{8,17,3}-\hat{y}_{8,17,3})^2}{\partial \hat{y}_{8,17,3}} \end{bmatrix}\right]\right] \\
&= \frac{1}{N*K*D}\left[\begin{bmatrix} 2(y_{1,1,1}-\hat{y}_{1,1,1}) & \dots & 2(y_{1,1,3}-\hat{y}_{1,1,3}) \\ \vdots & & \vdots \\ 2(y_{1,17,1}-\hat{y}_{1,17,1}) & \dots & 2(y_{1,17,3}-\hat{y}_{1,17,3}) \end{bmatrix};\right. \\
&\qquad \left.\left[\dots; \begin{bmatrix} 2(y_{8,1,1}-\hat{y}_{8,1,1}) & \dots & 2(y_{8,1,3}-\hat{y}_{8,1,3}) \\ \vdots & & \vdots \\ 2(y_{8,17,1}-\hat{y}_{8,17,1}) & \dots & 2(y_{8,17,3}-\hat{y}_{8,17,3}) \end{bmatrix}\right]\right] \\
&= -\frac{2}{N*K*D}(y-\hat{y})
\end{aligned}$$

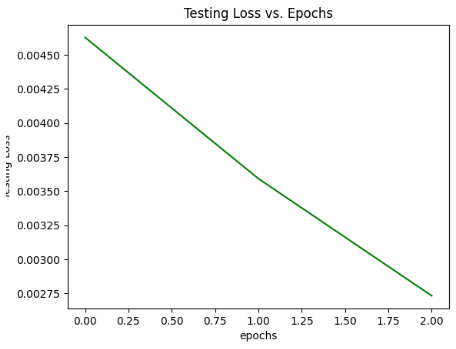
4 Results

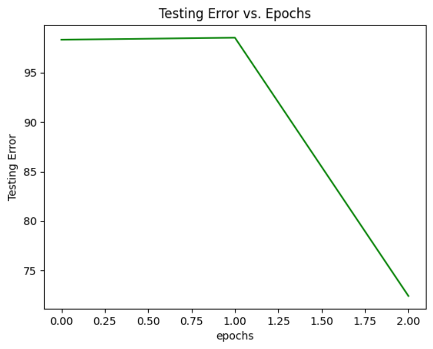
Training Loss and MPJPE plots:





Testing Loss and MPJPE plots:





Final MPJPE: 72.93 mm

5 Discussion

This model is built entirely using Tensorflow 2.8.0 and Numpy 1.22.2. It utilizes the `tf.keras.Model` parent class for each model and `tf.keras.layers` for building the layers of each model. The custom metric Mean Per Joint Position Error was built using the `tf.keras.metrics.Metric` class, while the loss function was pulled from `tf.keras.losses`.

The various hyperparameters of this model include the learning rate, the optimizer, and the training batch size. Varying hyperparameters was increasingly difficult to test for this model compared to previous models due to the much larger training time. For the Deep Dynamic Model, one epoch of training is about 35-40 minutes long, compared to 15 seconds for the Convolution Neural Network of the last assignment. Even though convergence was achieved at 3 epochs or less, total training still ended up taking around 2-3 hours when accounting for data preprocessing as well. Because of this, the final values of the hyperparameters weren't vigorously tested against other values. The final learning rate used was 0.0001, while the final optimizer used was the Adamax optimizer.

The batch size was unchanged for a different reason. The model was built and trained locally on a 2021 14-inch MacBook Pro with the Apple M1 Pro chip and 16GB RAM. These specs were enough to load the data, build and train the model, and make predictions, but increasing the batch size any further than the final value used led to an overflow of memory. For this reason, the only batch size used and tested was a batch size of 2.

Overall, the model trained successfully with a learning rate of 0.0001 on an Adamax optimizer, a Mean Squared Error loss function, and a batch size of 2 over 3 epochs. The loss would decrease significantly in the first epoch itself, but extra epochs reduced the loss by a factor of 7 and resulted in an even further improved error. The aggregated nature of the Deep Dynamic Model likely led to significant results within single iterations.