Sapienza University of Rome

Convolutional 3D
Body Movement Recognition

Ibis Prevedello, Jean-Pierre Richa

November 22, 2018

1 Introduction

The goal of this project was to train a model based on Convolutional 3D to recognize the human body movements in videos. To execute this task, the C3D network was used in combination with Openpose, which are coded using tensonflow. Openpose network was essential here, to extract the body pose, and apply it to the frame extracted from the video, then to train the C3D network to recognize the body motion.

The dataset used for the training was a subset of the Human 3.6M collection, which were taken in a local laboratory, and then labeled by the students who were taking the course of elective in Artificial Intelligence, which is a computer vision based elective course. The labeling was done using a json file, which was later split into training and testing set. In order to have a well balanced dataset for all of the classes, half of the classes that has the biggest amount of labels were considered for the training.

The training was done using a Google Compute Engine instance running a Tesla K80 GPU.

2 Dataset Augmentation

The labeled dataset contained in the json file was split into test set and training set, which are composed of 52 classes. Due to the difference of labeled samples between the classes, only those who had the highest numbers were considered for the training.

A total of 26 classes were taken into account to train the model, because the limited number of samples of the other classes provided a big chance of occur overfitting, and reduce generalization to other data. Check the **Classes section** for details about the used labels.

- Head
  - Turn right
  - Turn left
  - Raise
  - Lean forward
- Right/Left Arm
  - Shoulder extension
  - Shoulder adduction
  - Shoulder flexion
  - Shoulder abduction
  - Elbow flexion
  - Elbow extension
  - Roll the wrist
- Right/Left Leg
  - Hip flexion
  - Hip extension
  - Knee flexion
  - Knee extension

Since the data used was also not enough for the training to generalize, augmenting the training set was necessary. In order to maintain data consistency the following approach was applied; the idea was to flip the images labeled as arm and leg and add them to the opposite class, so for example the right shoulder extension becomes left shoulder extension and vice-versa. After performing these manipulations, the final dataset was extended to 400 samples for each class, with a total of 10,400 samples for the training set and 1900 samples for the test set, which enlarged the original dataset by approximately the third.

3 Body Movement Recognition

Body movements are too complex to be recognized, because when a person moves his/her head or any body part, only small variations can be seen in the features extracted. The main problem is that not only the specific part that the person intended to move is affected, but also other parts of the body, and this holds especially if the classifier should predict the movements in the real life, which is the case in this project. Even if the movement is small, it will affect the features extracted from other areas also, and each time with a different intensity, which makes it even harder to predict which part should be the main focus.

For these reasons, it was decided to use the help of openpose to extract a map of the current pose, which gives an accurate pose estimation of the body state, that should help the Convolutional 3D network to predict the body movement more accurately.

After estimating the pose of the body using the pre-trained model of openpose, the extracted maps were applied to the corresponding frames and then fed into the Convolutional 3D network using batches of 10 samples.

4 Network Training

The network was trained using transfer learning, which was achieved by employing the weights of the pre-trained model **sports1m finetuning ucf101**, used to classify sports activities. It was not easy to decide on a number of epochs that's enough to get a good accuracy, so after some testing it was discovered that the best to prevent overfitting and get the highest accuracy possible was 30 epochs, with a batch size of 10 samples.

Following the transfer learning approach, the last layer was deleted in order to exclude the final classification results of the previous model, and to be able to include the new total number of classes for this specific task. The training consisted of 2 phases, the first one was accomplished through 1st training only the last layer weights for 30% of the total number of epochs with a learning rate of $\mathbf{1} \times \mathbf{10^{-3}}$, 2nd training the weights of the whole network with a learning rate of $\mathbf{1} \times \mathbf{10^{-4}}$.

After each epoch, the updated weights were used to get the accuracy of the prediction made on the training set and the test set.

Two different loss functions were used to train the network and calculate the loss:

- **tf.nn.l2_loss:** the Least Squares Errors function, which was used inside the fully connected hidden layers, and its mathematical representation can be seen **below**

$$L2LossFunction = \sum_{i=1,n} (y - \hat{y})^2 \tag{1}$$

- **tf.nn.softmax_cross_entropy_with_logits:** the Cross Entropy loss function, which was used for calculating the loss on the output layer, then its gradient was calculated and back propagated through the network to update the weights. Its mathematical representation can be seen **below**.

$$H(y, \hat{y}) = -\sum_{i}(yi\ log\ \hat{yi}) \tag{2}$$

A summary that included the accuracy of training and testing set was evaluated after each epoch and can be visualized using Tensorboard.

5 Results

The network was trained for 30 epochs only, because it was enough to see how the accuracy got better with each epoch passed. It was best to stop the training after this number of epochs also, because the dataset is quite large, which makes the training take more time to improve the accuracy, and it wasn't possible to keep on training for more time, this was due to the fact that the training was done on the cloud and it's paid by hour.

After each epoch the accuracy of the model was saved using the summary collector, this way the accuracy can be checked using Tensorboard, which also gives the chance to extract the plot that is easily visualized **below**.

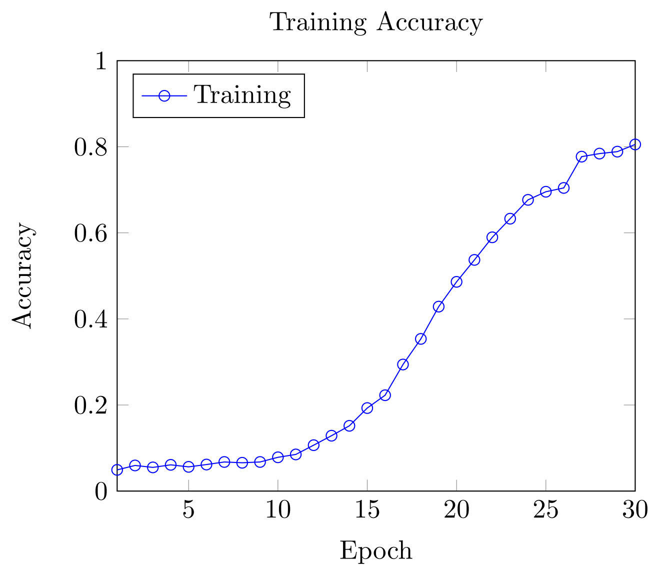

Figure 1: The plot shows how the training accuracy improves after each epoch.

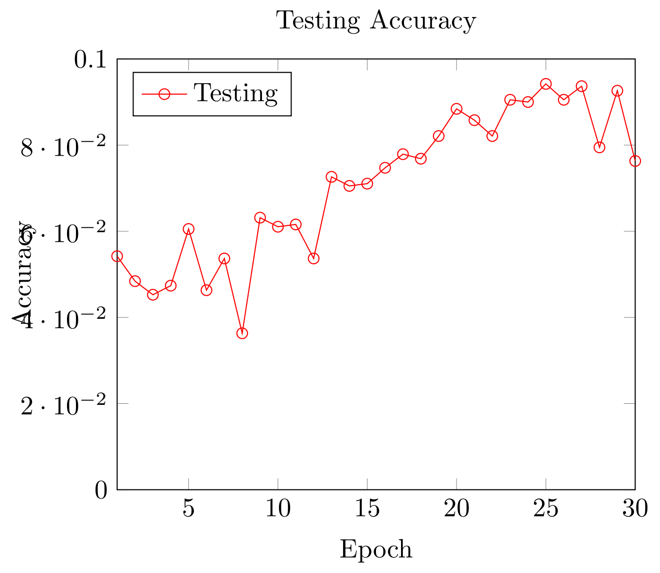

Figure 2: The plot shows how the testing accuracy improves after each epoch.

6 Implementation

The project has been implemented in Python using Tensorflow. Below, the list of Python files implemented for the project with a brief description of their behavior.

- **generate_tfrecords.py:** generating the tfrecord files used for the training and evaluation of the network.
- **train.py:** train the network specifying parameters and the dataset.
- **activities.py:** list of activities to be trained and the number of samples per activity to augment data.
- **c3d_model.py:** C3D model implementation.

Extra file:

- **pose_list.py:** shows the list of activities and the frequency of activities chosen to the training.

7 Conclusion

As already shown in the report, the accuracy achieved on the training set is 80%, and on the testing set was 8%, this is a sign of overfitting, even though all the steps to prevent this behavior were taken. It is obvious that the accuracy on the evaluation set was increasing, but there are a lot of obstacles that kept the network from achieving better results. The training set was not enough for the network to generalize to other samples, because with this number of classes, a wider set was needed in order to get a higher accuracy. Moreover, the improvement in the accuracy over the epochs was good with respect to the number of classes to be classified, so if the network can be trained for more epochs, the accuracy on both sets would be higher.