Sai Surya Nattuva

Prof. Gokhan Kul

DSC 550

University of Massachusetts Dartmouth

Human Activity Classification from Time-Series Data

Abstract

Human activity can be observed and measured by employing different sensors on different parts of the body. An increase in the usage of wearable devices by people has led to increased research in Human Activity monitoring and analysis. This analysis has helped develop many technologies like fall detection, irregular heart rhythms, walking steadiness, posture, etc. This project focuses on classifying the activity performed by a human from time-series data generated from the user's phone's accelerometer. The time-series data is first converted into 2-D images and then fed to a CNN to learn five different activities from 4 features of the data. The project's primary focus is to leverage the predicting power of CNNs on time-series data with a small number of training samples and input channels.

Data Collection

The project focuses on the classification of the different punches thrown by the user during a practice session of boxing. In the experimental setup, the user held an android smartphone in his dominant hand (right hand in this case). The movements were recorded using an app called “Physics Toolbox Sensor Suite Pro". The app keeps recording the movements of the smartphone while the user is performing his routine. The app collects data from the accelerometer of the device which includes the X, Y, Z axes, and the G-force. The data is polled at 100Hz. We have recorded a practice boxing session for 77 seconds for this project. The data is

generated with the time stamp and the values of the above measurements in a CSV format. The columns in the CSV are (in order) – Time, X, Y, Z, G-force. Since the app saves the time in "ss.mmmmmm" (seconds.microseconds) format, I have formatted the timestamps to GMT standard format using a custom Python script, to help with the next step.

Table 1

The features available data are described as below.

Feature	Measurement
TS	Timestamp of the data in the format (YYYY-MM-DD hh:mm:ss.mmmmmm)
gFx	Linear acceleration in the X-axis
gFy	Linear acceleration in the Y-axis
gFz	Linear acceleration in the Z-axis
TgF	G-force of the device

Fig 1 Time series data with 4 modalities – gFx, gFy, gFz, TgF on the Y-axis and the elapsed time on the X-axis

Data Labeling

Since the data generated by the smartphone is unlabeled, I have used an open-source data labeling tool "Label-Studio" to label the data. Label-Studio is a graphical user interface where we can drag along the timeline of the data and label those regions appropriately.

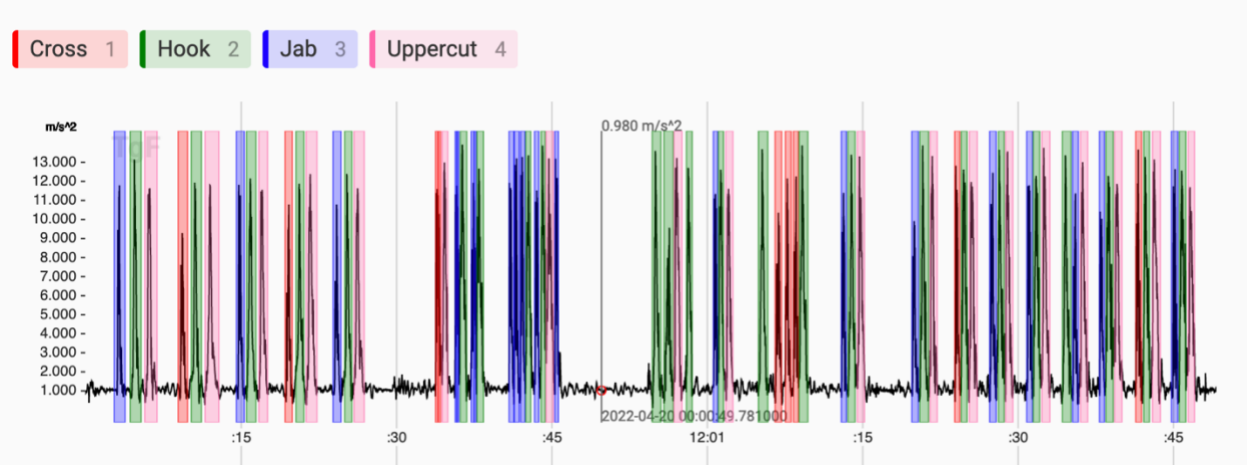

Fig 2 Label Studio interface snippet

Label-Studio can take a CSV as input time-series data that can be added as a labeling project in the application. Once we finish the labeling tasks, we can export the annotations in the form of a JSON file. The JSON file contains the annotations as an array of labels assigned to regions of the time-series data denoted by the start and stop times. To add the labels to all the

data points in the data, I have written a Python script to read the regions from the JSON file and assign the labels to the appropriate data points in those regions. I have assigned a constant label for the rest of the unlabeled regions of the time-series data. In this way, every data point in the time series has a label associated with it.

Creating 2-D Images

We are using 4 features namely - gFx, gFy, gFz, and TgF, of the data to train the Neural Network model. The raw stream of data is first normalized column-wise using the *l2* norm to normalize the scale of the data. The normalized data is converted into 2-D images by sliding a 64-size window through a transposed representation of the data. The window is slid across the data with a span size of 32. This step effectively generates 64-size 2-D representations of the data which are of the shape (4, 64) where 4 is the number of features and 64 is the size of the window. Since each image has only 1 label associated with it, the label at the end time stamp of the image window is considered the label for the image. This process created 330 images from the available time-series data.

Neural Network Architecture

The Deep Neural Network architecture that I have used to build the model is adapted from the research paper “SensorNet: A Scalable and Low-Power Deep Convolutional Neural Network for Multimodal Data Classification”. This architecture is optimized to run smoothly on low-power devices like SoCs, or embedded systems. The NN architecture consists of 5 convolutional layers, 3 pooling layers, and 1 fully connected dense layer. The output layer is a Softmax layer that predicts the class of the passed input images.

The filters in the convolutional layers generate the shared features between the different dimensions or modalities of the data. Since it is assumed that there is no spatial correlation

among the different modalities of the data, a filter of size (4, 5) is used in the first convolutional layer, and filters of size (1, 5) are used in the subsequent convolutional layers. To pass these learned features to the dense layer, there are 3 max-pooling layers that flatten the image with a pool size of (1, 2) after the second, fourth, and fifth convolutional layers. This flattened image is then passed to the final 2 fully connected layers of the network. First is a 64-node dense layer, which is fully connected to a 5-node (as we have 5 output classes) Softmax output layer that gives the probability scores for the different output classes.

I have chosen the “ReLU” activation function, learning rate of 0.001, the “Adam” optimizer and the “Sparse Categorical Crossentropy” as the loss function for the NN layers based on the results of hyperparameter tuning, which is explained in a further section.

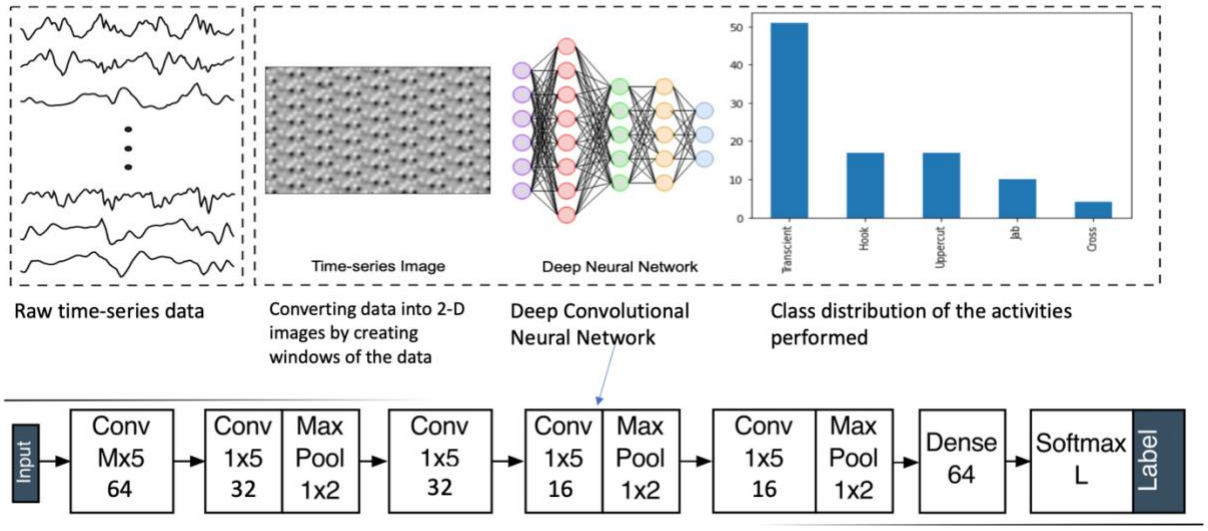

Fig 3 SensorNet Architecture

Table 2

This table contains the details of the architecture of the Neural Network.

Layer	Configuration
Conv Mx5 64	64 node convolutional layer
	4x5 filter size
Conv 1x5 32	32 node convolutional layer
	1x5 filter size
MaxPool 1x2	Max pooling layer
	Pool size (1, 2)
Conv 1x5 32	32 node convolutional layer
	1x5 filter size
Conv 1x5 16	16 node convolutional layer
	1x5 filter size
MaxPool 1x2	Max pooling layer
	Pool size (1, 2)
Conv 1x5 16	16 node convolutional layer

	1x5 filter size
MaxPool 1x2	Max pooling layer
	Pool size (1, 2)
Dense 64	64-node fully connected layer
Softmax L	5-node Softmax output layer

Model Pipeline

Now that we have described the input processing and NN architecture, the model pipeline is described in this section. The pipeline consists of 4 steps namely – input processing, data segmentation, model training, and model performance evaluation. The steps are described briefly below.

Input Processing – This step is where the normal input time series data is converted into 2-D images along with the corresponding labels.

Data Segmentation – This step is where we segment the data into training, validation, and testing sets. The proportions of the training, validation and testing data sets are 55%, 15%, and 30% respectively. The segmentation is done in the chronological order of the input images.

Model Training – This step is where we train the Neural Network model on the training data and then validate the learnings of the model with the validation data. The model I have used for this architecture is a Keras Sequential model, stacked with the layers as mentioned in the "Neural Network Architecture" section. The model is compiled with the "Adam" optimizer and a

learning rate of 0.001. The model is trained with a batch size of 32 for 150 epochs and the loss function is “Sparse Categorical Crossentropy”.

Model Performance Evaluation – This step is where we test the model performance by predicting the labels for the testing data. I have considered the F-1 Score and the ROC-AUC plots as the basis for measuring the performance of the model.

Hyperparameter Tuning

After the model pipeline was setup, I have trained the hyper parameters for the NN by making a grid of parameters to optimize by utilizing the “RandomizedSearchCV” algorithm from “SKLearn” Python module with 3 cross-fold validation. The “accuracy score” is used as the measure for optimizing the hyperparameters. The parameters that I have selected for tuning are as below.

Table 3

This tables lists out the hyperparameters optimized for the Neural Network using RandomizedSearchCV

Tuned Parameter	**Range**	**Optimized Value**
Number of filters	[16, 32, 64, 128]	**64**
Activation function	[ReLU, Tanh]	**ReLU**
Epochs	[50, 100, 150, 200]	**150**
Batch size	[16, 32, 64, 128]	**32**

Loss function	["CategoricalCrossEntropy", "SparseCategoricalCrossEntropy"]	"**SparseCategoricalCrossEntropy**"
Optimizer	["Adam", "RMSProp", "SGD"]	**"Adam"**
Learning Rate	[0.1, 0.01, 0.001, 0.0001]	**0.001**

Model Performance

The model performance is tested by evaluating the predictions of the model on the testing data. I have used "F-1 score" and "ROC-AUC" score and the "ROC" curves as the means to measure the model performance.

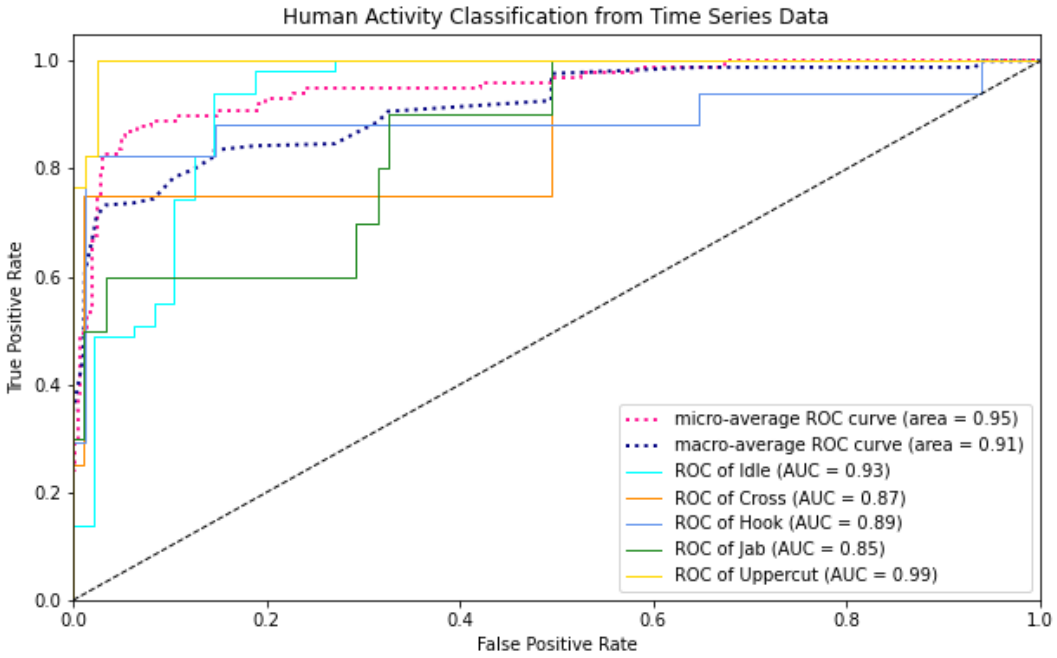

Fig 4 ROC-AUC curves for the output labels

Table 4

The following table lists out the ROC-AUC accuracies and the F-1 scores of the model.

Output Label	ROC-AUC (Higher is better)	F-1 Score (Higher is better)
Idle	0.93	0.88
Cross	0.87	0.33
Hook	0.89	0.87
Jab	0.85	0.56
Uppercut	0.99	0.89
Overall	**0.95 (micro-average)**	**0.84**

The classification report for the testing data is as below.

	precision	recall	f1-score	support
Idle	0.87	0.90	0.88	51
Cross	0.50	0.25	0.33	4
Hook	0.93	0.82	0.87	17
Jab	0.62	0.50	0.56	10
Uppercut	0.81	1.00	0.89	17
accuracy			0.84	99
macro avg	0.75	0.70	0.71	99
weighted avg	0.83	0.84	0.83	99

With these metrics, we can conclude that the model's prediction accuracy is 95% (based on the micro-averaged ROC-AUC score) in predicting the labels.

Works Cited

A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates and T. Mohsenin, "SensorNet: A Scalable and Low-Power Deep Convolutional Neural Network for Multimodal Data Classification," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 1, pp. 274-287, Jan. 2019, doi: 10.1109/TCSI.2018.2848647.

Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

M. Tkachenko, M. Malyuk, A. Holmanyuk, N. Liubimov, Label Studio: Data labeling software https://github.com/heartexlabs/label-studio, 2020-2022

Vieyra Software, Physics Toolbox Sensor Suite Pro, https://play.google.com/store/apps/details?id=com.chrystianvieyra.physicstoolboxsuite&gl=US