**Authors:** Attia Hamouda **Date:** 28-05-2023

# **Title:** Gym exercice classification model using Flux in Julia

## 1. Introduction

The purpose of this report is to provide an overview of the code and steps involved in a exercise classification task using neural networks. The code utilizes the Flux and MLJ packages in Julia to preprocess the data, train a neural network model, and evaluate its performance. The goal is to predict the target body part based on the given exercise name.

## 2. Data Preprocessing

The code begins by reading a CSV file **("megaGymDataset.csv")** containing exercise names and their corresponding target body parts into a DataFrame using the CSV and DataFrames packages. The DataFrame allows for easy manipulation and analysis of the data.

Summary statistics of the DataFrame are then displayed using the describe() function, providing insights into the data's structure and distribution. The number of rows and columns in the DataFrame are obtained using the nrow() and ncol() functions, respectively.

```
[167]: # Read the CSV file into a DataFrame
       df = CSV.read("megaGymDataset.csv", DataFrame)

       # Display summary statistics of the DataFrame
       describe(df)

       # Get the number of rows and columns in the DataFrame
       nrow(df), ncol(df)

[167]: (2918, 9)
```

Next, the code selects the two columns of interest, **"Title"** (exercise names) and **"BodyPart"** (target body parts), from the original DataFrame. A new DataFrame, new_df, is created with the selected columns.

```
[5]: # Select the two columns from the original DataFrame
     selected_columns = df[:, [:Title,  :BodyPart]]
     # Create a new DataFrame with the selected columns
     new_df = DataFrame(selected_columns)
     nrow(new_df), ncol(new_df)

[5]: (2918, 2)
```

The unique() function is used to remove any duplicate rows in new_df, and the row indices of the DataFrame are reset.

## 3. Model Training

The code proceeds with the feature extraction step by one-hot encoding the exercise names (X) using Flux.onehotbatch(). The resulting encoded matrix, X_encoded, is of type Matrix{Float32}.

The target labels (body parts) are encoded as integers using a label mapping. Each unique body part is assigned an integer value using the Dict() function. The target labels are then encoded into y_encoded using the label mapping.

The data is split into training and testing sets using a specified train-test split ratio. The data is randomly shuffled using the shuffle() function from the Random package. The sizes of the training and test sets are calculated based on the split ratio.

The neural network model architecture is defined using the Flux package. It consists of three dense layers with a ReLU activation function. The input size is determined based on the number of unique exercise names.

```
[14]: # Define the model architecture
input_size = size(X_encoded, 2)
model = Chain(
    Dense(input_size, 64, relu),
    Dense(64, 32, relu),
    Dense(32, length(label_mapping))
)
```

```
[14]: Chain(
        Dense(2909 => 64, relu),        # 186_240 parameters
        Dense(64 => 32, relu),          # 2_080 parameters
        Dense(32 => 17),                # 561 parameters
      )                 # Total: 6 arrays, 188_881 parameters, 738.191 KiB.
```

A loss function is defined using Flux.crossentropy() to calculate the cross-entropy loss between the predicted and true labels. An Adam optimizer is utilized for model optimization.

```
[15]: # Define the loss function
loss(x, y) = Flux.crossentropy(softmax(model(x)), Flux.onehotbatch(y, 1:length(label_mapping)))
# Define the optimizer
optimizer = Flux.ADAM()
```

```
[15]: Adam(0.001, (0.9, 0.999), 1.0e-8, IdDict{Any, Any}())
```

The model is trained for a specified number of epochs using the Flux.train!() function, passing the loss function, model parameters, training data, and optimizer. The model parameters are updated iteratively to minimize the loss.

### 4. Model Evaluation

After training, the code proceeds with model evaluation. The test set is used to make predictions using the trained model. Exercise names (X_test) and true body parts (y_test) are extracted from the test data.

The argmax() function from Flux is used to determine the predicted body parts (y_pred) by finding the index of the maximum value in the model's output probabilities.

The accuracy is calculated by comparing the predicted body parts with the true body parts and calculating the proportion of correct predictions. The accuracy value provides an indication of the model's performance on the test set.

```
[17]: # Step 4: Model Evaluation
# Make predictions on the test set
X_test = [x for (x, _) in test_data]
y_test = [y for (_, y) in test_data]
y_pred = Flux.argmax(model.(X_test), dims=2)
# Calculate accuracy
accuracy = sum(y_pred .== reshape(y_test, :)') / length(y_test)

println("Accuracy: $accuracy")
```

```
Accuracy: 1.0
```

Finally, an example exercise name ("Band low-to-high twist") is selected for prediction. The exercise name is one-hot encoded and passed through the trained model to obtain the predicted body part. The predicted body part is decoded using the inverse label mapping, providing the final predicted body part label.

## 5. Conclusion

In conclusion, this code demonstrates a workflow for exercise classification using neural networks. The code efficiently preprocesses the data, trains a neural network model, and evaluates its performance. The achieved accuracy on the test set provides insights into the model's predictive capabilities. The code can be further expanded and modified to accommodate different datasets and experimentation with different model architectures, hyperparameters, and evaluation metrics.