MoBeats

AN INTELLIGENT VISION-TRANSFORMER GAME SYSTEM

Pattern Recognition Systems
Practice Module
Semester II 2021/2022

Prepared by:

Chiu Man Shan
Zhao Lutong
Kuch Swee Cheng
Chen Hao

1. CONTENTS

2. EXECUTIVE SUMMARY

MoBeats seeks to provide vision-based user inputs into computer games. The appeal and focus of vision-based brings immersive and interactive into application such as rehabilitation exercise by improving motor ability in stroke patient through game-assisted therapy sessions. Computer vision classification algorithm are used in such human-game interactions for accurate gesture recognition in real-time. In this intelligent systems project, we seek exploit advances in deep convolutional network methods for efficient image filtering and explore Vision Transformer(ViT), LSTM and RNN model for predicting sequence of temporal action for gesture pattern recognition.

3. PROBLEM DESCRIPTION

3.1 STATEMENT

Upper-limb impairments suffered by stroke patients could severely affect their quality of lift. Scientific evidence support that repetitive rehabilitation exercise can improve motor ability and aid faster recovery in stroke patient. Computer vision based rehabilitation games has the tremendous potential to make mundane and repetitive sets of rehabilitation therapies more engaging and improve adoption rate by stroke patient.

MoBeats seeks to incorporate computer vision-based algorithm as game input to rhythmic game as real-time game-assisted rehabilitation therapy. The need for high speed and low cost requirement for such remote rehabilitation games requires the pose classification model to be pre-loaded onto a website using a laptop camera as vision-based input.

3.2 OBJECTIVE

Provide a real-time and low cost game using computer vision-based algorithm accessed over a website for remote game-assisted rehabilitation therapy.

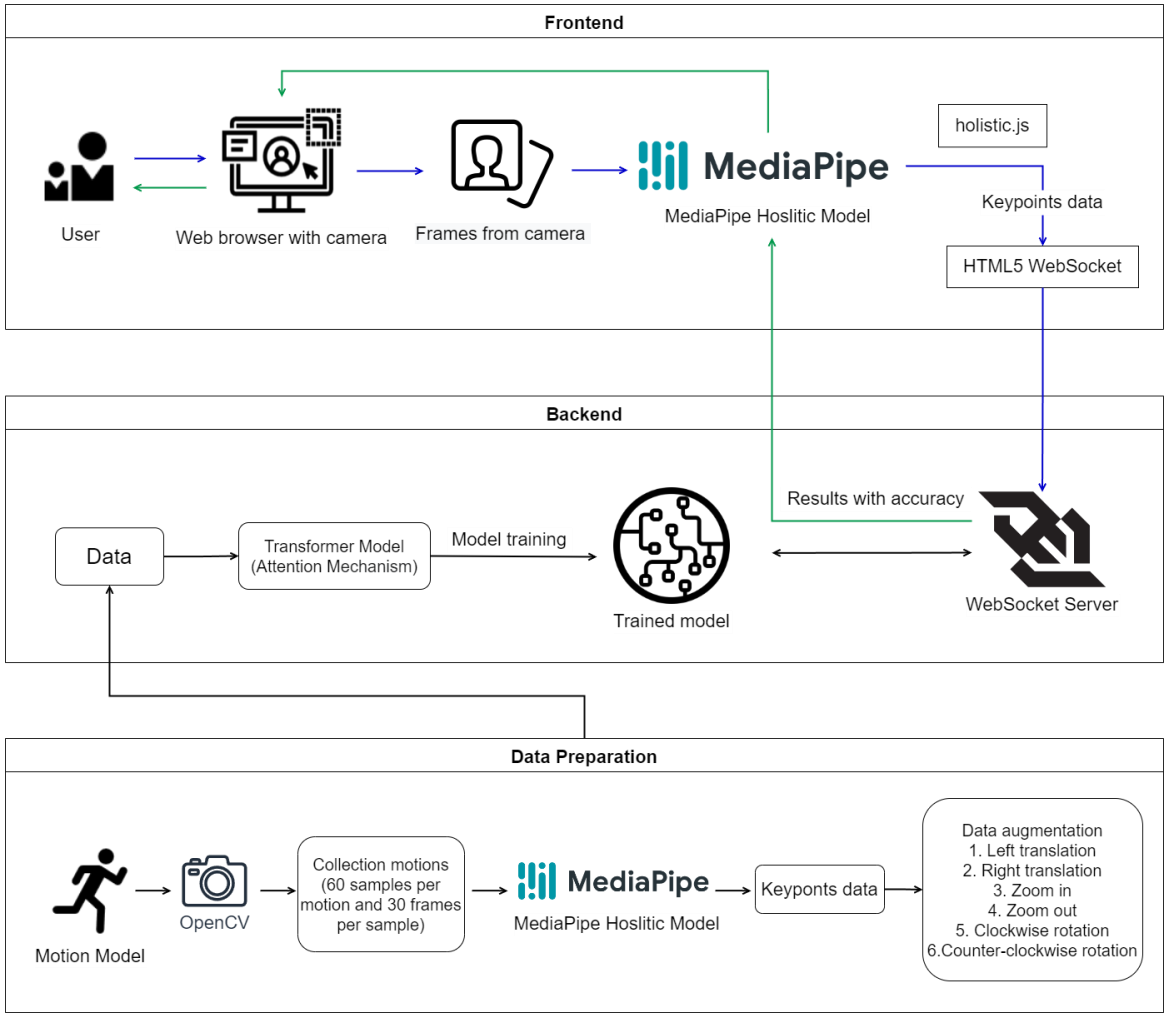
Achieve a set of upper-limb temporal therapy action that requires reaching movement in a rhythmic musical game which patient need to follow. Upper-limb keypoint detection and neural network training are required for pose estimation for recognition. Pre-processing and feature extraction are then performed on the image for image classification. Transformer is used in real-time pose classification which translate to input for the game. A prediction score on the accuracy of the action and classification result is displayed in real-time which translates as the input to the rhythmic game.

4. SYSTEM DESIGN AND MODEL

4.1 OVERALL SYSTEM DESIGN ARCHITECTURE FOR MOBEATS

Our proposed solution is deployed on a web browser with an RGB laptop camera to identify the posture of the user. Camera grabs the video streaming data by image and passes the frame into MediaPipe Holistic model, which output the keypoint and landmark data of the user. These data are parse to the pre-trained model through the WebSocket server which listens for the classified pose recognized from the model.

The Transformer model is pre-trained with custom labelled action dataset acquired through OpenCV. Images are pre-processed and augmented(Scaling, Translation and Rotation) to better generalize the recognizer model in real-time deployment making the model more robust. The Transformer model would predict and classify the acquired images from the user to generate the most probable pose which results in a pre-defined input to the rhythmic game.



4.2 DATA COLLECTION AND PRE-PROCESSING

In this intelligent systems project, a custom image dataset is required for pretraining of the recognizer model. To minimize the impact from different background and improve the recognition of the pose, keypoint extract method is chosen to extract the landmark information require to predict and classify the pose. The main libraries used for data collection are MediaPipe Holistic Model and OpenCV.

OpenCV does the initialization and necessary conversion of images frame from the video stream data from the webcam from BGR to RGB. Image frames are then passed into MediaPipe Holistic pipeline which is an integration of three separate models consisting of pose, face and hand components. Each of the component are recorded to ensure the flexibility and generalization of the model during the model training phase.

Using MediaPipe Holistic model, landmark information from pose, face, hand components are generated creating an array of the prediction classes for each landmark. The list of landmark vector NumPy array for each prediction classes are passed to the deep learning network for training.

For the points out the frame, they will be set to zero to keep the consistent data shape for the following model training. Hence, the recorded NumPy data file for each frame consists of 1662 data (i.e. 33×4 for pose landmarks, 468×3 for face landmarks and 21×3 for each left and right hand landmarks).

To ensure a sufficient and balanced labelled image data is used for model training, we scripted the data collection four different sets of training pose (i.e Left Punch, Right Punch, Wakanda and Kiss) for 60 times with 30 frames inside a one-time motion. A no-class action was also created and trained with random pose using the data collection script. This serves as a buffer class for transient movement which does not fall into any of the intended training pose avoiding misclassification.

4.3 DATA AUGMENTATION

Due to the limited data size of the recorded keypoint, our team chose to carry out the data augmentation to increase the size of data used for training a model. The collected data is augmented in order to make for a better generalized and robust model. After learning from the common data augmentation methods on the image dataset, adopted the following position augmentation techniques to adjust the coordinates of the keypoint:

1. Scaling: the x and y coordinates will be adjusted based on the zoom in/out from the centre point and below are the functions for each treatment:

```
def zoom_in(x,y,ratio):
 return 0.5+(x-0.5)*ratio,0.5+(y-0.5)*ratio

def zoom_out(x,y,ratio):
 return 0.5+(x-0.5)/ratio,0.5+(y-0.5)/ratio
```

2. Translation: the x coordinates will be adjusted based on the left / right translation method and below are the functions for each treatment:

```
def left(x,y,ratio):
    return x-ratio,y

def right(x,y,ratio):
    return x+ratio,y
```

3. Rotation: the x and y coordinates will be adjusted based on the clockwise /counter-clockwise for 15° from the centre point and below are the functions for each treatment:

```
def cw(x,y,ratio):
    ratio = ratio/180*math.pi
    r = ((x-0.5)**2 + (y-0.5)**2)**0.5
    xita = math.atan((y-0.5)/(x-0.5))
    alfa = xita - ratio
    return 0.5+r*math.cos(alfa),0.5+r*math.sin(alfa)

def ccw(x,y,ratio):
    ratio = ratio/180*math.pi
    r = ((x-0.5)**2 + (y-0.5)**2)**0.5
    xita = math.atan((y-0.5)/(x-0.5))
    alfa = xita + ratio
    return 0.5+r*math.cos(alfa),0.5+r*math.sin(alfa)
```

4.4 TECHNIQUES

4.4.1 MEDIAPIPE MODEL

Adopting MediaPipe Holistic model is a winning strategy in terms of speed, and flexibility it brings to the overall system design consideration. Using MediaPipe Holistic, as a pose recognition module this allows use to capture the user movement and passed the classification result to translate into game based input.

MediaPipe Holistic is an integrated pipeline which incorporate three separate models consisting of pose, face and hand component. The model generates the keypoint and landmarks data vector which is fed into the transformer model for deep learning training and classification of the pose.

From the output from MediaPipe detection, the following list of keypoint and landmarks will be extracted:

- POSE_LANDMARKS

  Each landmark consists of the following:

1. x and y: Landmark coordinates normalized to[0.0, 1.0]by the image width and height respectively.
2. z: Should be discarded as currently the model is not fully trained to predict depth, but this is something on the roadmap.
3. visibility: A value in [0.0, 1.0] indicating the likelihood of the landmark being visible (present and not occluded) in the image.

- FACE_LANDMARKS

  A list of 468 face landmarks. Each landmark consists of x, y and z. x and y are normalized to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at center of the head being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

- LEFT_HAND_LANDMARKS

  A list of 21 hand landmarks on the left hand. Each landmark consists of x, y and z. x and y are normalized to [0.0, 1.0] by the image width and height respectively. z represents the landmark depth with the depth at the wrist being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

- RIGHT_HAND_LANDMARKS

  A list of 21 hand landmarks on the right hand, in the same representation as left_hand_landmarks.

4.4.2 TRANSFORMER (ATTENTION) MODEL

To achieve a swift and responsive video data processing in real-time, we explore transformer model a deep learning model that adopts the mechanism of self-attention for sequential processing. Compared to its predecessor the RNN network, which faces the vanishing gradient problem, where the lower the gradient the harder it is for the network to continually train and update the weight creating a domino effect on further weights propagating through the network. This leads to a longer processing time of updating the weight in the neural network.

Transformer model is a deep learning model which comes with additional attention mechanism which was primarily used in the subfield of natural language processing(NLP). The transformer learns by measuring the relationship between input token pairs that occurs in sequential data by accessing all previous states and weight according to a learned measure of relevance.

useful in our Computer Vision application. Unlike its RNN and LSTM predecessor, transformer model process the entire input all at once allowing for more parallelization and therefore reducing training and processing time.

MODEL SUMMARY

Layer (type)	Output Shape	Param #	Connected to
input_9 (InputLayer)	[(None, 6, 258)]	0	[]
layer_normalization_40 (LayerNormalization)	(None, 6, 258)	516	['input_9[0][0]']
multi_head_attention_20 (MultiHeadAttention)	(None, 6, 258)	265218	['layer_normalization_40[0][0]', 'layer_normalization_40[0][0]']
dropout_48 (Dropout)	(None, 6, 258)	0	['multi_head_attention_20[0][0]']
tf.__operators__.add_40 (TFOpLambda)	(None, 6, 258)	0	['dropout_48[0][0]', 'input_9[0][0]']
layer_normalization_41 (LayerNormalization)	(None, 6, 258)	516	['tf.__operators__.add_40[0][0]']
conv1d_40 (Conv1D)	(None, 6, 2)	518	['layer_normalization_41[0][0]']
dropout_49 (Dropout)	(None, 6, 2)	0	['conv1d_40[0][0]']
conv1d_41 (Conv1D)	(None, 6, 258)	774	['dropout_49[0][0]']
tf.__operators__.add_41 (TFOpLambda)	(None, 6, 258)	0	['conv1d_41[0][0]', 'tf.__operators__.add_40[0][0]']
layer_normalization_42 (LayerNormalization)	(None, 6, 258)	516	['tf.__operators__.add_41[0][0]']
multi_head_attention_21 (MultiHeadAttention)	(None, 6, 258)	265218	['layer_normalization_42[0][0]', 'layer_normalization_42[0][0]']
dropout_50 (Dropout)	(None, 6, 258)	0	['multi_head_attention_21[0][0]']
tf.__operators__.add_42 (TFOpLambda)	(None, 6, 258)	0	['dropout_50[0][0]', 'tf.__operators__.add_41[0][0]']
layer_normalization_43 (LayerNormalization)	(None, 6, 258)	516	['tf.__operators__.add_42[0][0]']
conv1d_42 (Conv1D)	(None, 6, 2)	518	['layer_normalization_43[0][0]']
dropout_51 (Dropout)	(None, 6, 2)	0	['conv1d_42[0][0]']
conv1d_43 (Conv1D)	(None, 6, 258)	774	['dropout_51[0][0]']
tf.__operators__.add_43 (TFOpLambda)	(None, 6, 258)	0	['conv1d_43[0][0]', 'tf.__operators__.add_42[0][0]']
global_average_pooling1d_8 (GlobalAveragePooling1D)	(None, 6)	0	['tf.__operators__.add_43[0][0]']
dense_16 (Dense)	(None, 128)	896	['global_average_pooling1d_8[0][0]']
dropout_52 (Dropout)	(None, 128)	0	['dense_16[0][0]']
dense_17 (Dense)	(None, 5)	645	['dropout_52[0][0]']

Total params: 536,625
Trainable params: 536,625
Non-trainable params: 0

HYPER PARAMETERS

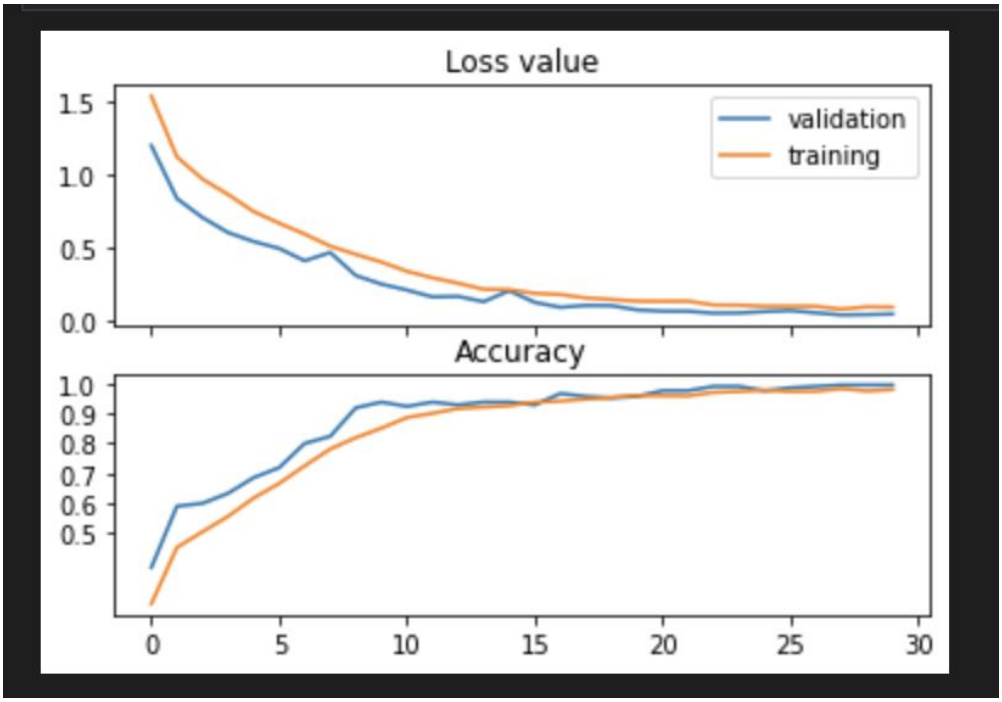
Loss function: **categorical_crossentropy**
Optimizer: **Adam**
No. of encoder block = 2
Head size = 128
Learning rate = 1e-4
Dropout = 0.1
Batch size = 40

MOTIONS

Left punch
Right punch
Kiss
No action

OVERALL PERFORMANCE

Loss: 0.046381287276744 84
Accuracy: **0.9952380657196045**



Average prediction time in 100 samples: 76ms, min: 70ms, max: 144ms

```
--- Average prediction time in 100 samples : 76ms, min: 70ms, max: 144ms ---
```

4.4.3 HYBRID MACHINE LEARNING

MediaPipe Holistic is an integrated pipeline which incorporate three separate model consisting of body pose, face and hand component to be used in this project. The model generates the keypoint and landmarks data vector which is fed into the transformer model for deep learning training and real-time classification achieving Hybrid Machine Learning approach.

4.5 INTELLIGENT SENSE MAKING

4.5.1 VIDEO CAPTURING

Using OpenCV(Open Source Computer Vision Library), video stream data is acquired through laptop camera which allows for camera module initialization in typical while-true cycle. Image frames are then grabbed and passed to the image processing module in MediaPipe for further stages of conversion and processing.

4.5.2 GAME BASED INPUTS

To input a command into the game, each gesture should represent a command for the game while creating a no-class action as a buffer for transient movement which does not fall into any of the pre-defined gesture. This helps to remove glitches or inconsistent recognition which decrease the possibility of transient actions being misclassified and registered as an input to the game.
System performance

5. PROJECT FINDINGS AND DISCUSSION

5.1 PROJECT FINDINGS – MODEL EXPLORATION

5.1.1 MODEL 1 – CNN TRANSFER LEARNING MODELS WITH NO KEYPOINT

In our early exploration stage, we firstly experimented with models with key-point detection and image-wise classification. Below are the parts for conducting image-wise classification:

We have experimented with popular image recognition algorithm which includes VGG16, VGG19 and InceptionV3 models, they share common code structure as follows:

- We firstly downloaded a pretrained model from ImageNet, exclude the fully connected layers.
- Then we will freeze all the layers in this model and attach dense layers of our own to be trained. In the last layer, we will add SoftMax non-linearity to get probability result for each class.
- We then train this model on our own dataset with Adam optimizer and recorded down the evaluation metrics.

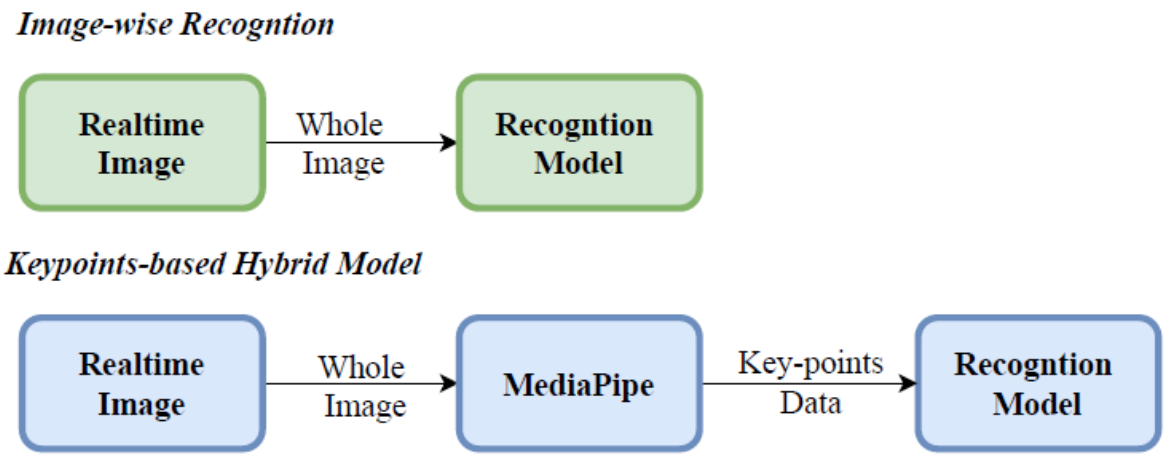
```
# Import model with no FC layers
vgg16 = VGG16(input_shape = img_size + [3], weights='imagenet', include_top=False)
```

```
# Freeze all layers
for layer in vgg16.layers:
  layer.trainable = False
```

```
# our dense layers
x = Flatten()(vgg16.output)
x = Dense(256, activation='relu')(x)
prediction = Dense(len(num_classes), activation='softmax')(x)
```

```
# creat model object
model = Model(inputs = vgg16.input, outputs = prediction)
model.summary()
```

After model exploration and comparison between the evaluation, we concluded that that using keypoint models leads to a more robust model which have greater generalization ability. Therefore, we decided to configure our model architecture



with a two stage hybrid model approach which incorporates keypoint and landmark data as vector into the recognition model for classification :

5.1.2 MODEL 2 – MACHINE LEARNING MODELS WITH KEYPOINTS

After the first stage, we decided to use a two-stage hybrid model. We firstly explored several traditional machine learning models on MediaPipe output. We firstly used KNN as our baseline model, then experimented with 2 ensemble models (1 boosting and 1 bagging). Lastly we tried a dimension reduction model LDA (Linear discriminant analysis).

After we trained on ensemble models, the performance for ensemble models is quite disappoint. We suspected the dimension may be too high such that it causes too much noise for model to converge. So we tried two ways to possibly increase the model performance.

Dimension Reduction

The 1662 data points represent 33*4 (body) + 468*3 (face) + 21*3 (Left Hand) + 21*3 (Right Hand). Since the main distinguish not emphasizing on face emotions so I removed the face data. Therefore it reduced the dimension from 1662 to 258 significantly.

As we can see from the two tables below. This dimension reduction did not contribute significantly to model accuracy improvement for both KNN and ensemble models.

(0.001 and 0.02 increase). However, it largely reduced the inference time for KNN from 0.2 to 0.05s.

Train Data Augmentation

The original dataset has a train and test data with shape (54000,1662). I firstly moved half of the data from train to test. Therefore, it become (81000,1662) and (27000,1662) for train and test data. This augmentation on train data has a very significant contribution for all the data. Especially random forest, which increased from 0.55 to 0.93.

However, the inference time for KNN also increased from 0.2 to 0.3s. This is because the model are comparing more data (more calculation) when deciding which is the nearest neighbour.

Model Test Accuracy	*Original Dataset*	*After Augmentation*	*After Augmentation And face-removal*
KNN	0.8602592592592593	0.9384814814814815	0.9398148148148148
Random Forest	0.5513703703703704	0.9312592592592592	0.9546666666666667
Ada Boosting	0.5757037037037037	-	0.7946296296296296
LDA	-	-	0.9548148148148148

Model Prediction Time	*Original Dataset*	*After Augmentation*	*After Augmentation And face removal*
KNN	0.20498871803283691	0.3129016160964966	0.05199146270751953
Random Forest	0.03656156063079834	0.0446373462677002	0.04009609222412109
Ada Boosting	0.00708682537078857	-	0.0488757848739624
LDA	-	-	"0.0"

It is interesting to note that LDA as a dimension reduction model has very low inference time and high accuracy for this task. This is not a popular model but delivers a surprising result with comparable performance (0.954) and very short prediction time.

However, although the machine learning model could reach 0.9548, we think we could achieve even better performance with use of the sequence data. For these four models we treat all frames as individual input. But for interactive game, the users

input are better recognized as motion than static pose. Therefore, we would like to explore models which capture sequential information as the next stage.

5.1.3 MODEL 3 – LSTM MODEL

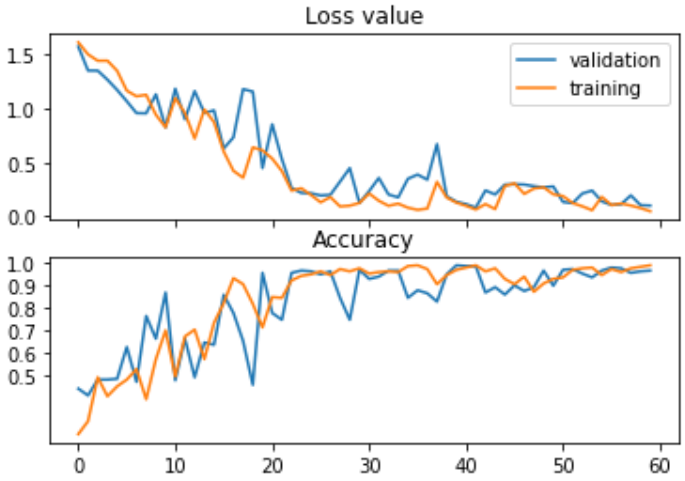
In recent years, deep learning represented by recurrent neural networks (RNNs) has become popular in processing variable length sequences of inputs. Long short-term memory (LSTM) neural networks is one of the most widely used gated RNNs. LSTM has the memory capacity to process time series. Motion detection can be regarded as a time series classification problem. Accordingly, the identification based on LSTM has shown promising performance.
There are 6 hidden layers behind the LSTM layer, and finally a fully connected layer with an output dimension of 5.

Key considerations of adding hidden layers can directly enhance the performance of the model by providing a greater range of feature representation. However, more hidden layers is not better when it comes to real-time deployment. Excessive hidden layers will bring issues such as difficulty in convergence, overfitting and excessive back propagation of weights in the neural network. Therefore, it is very important to choose the appropriate number of hidden layers in the neural network, that provides optimal performance to the LSTM network model.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm_9 (LSTM)                (None, 30, 64)            442112

 lstm_10 (LSTM)               (None, 30, 128)           98816

 lstm_11 (LSTM)               (None, 30, 256)           394240

 lstm_12 (LSTM)               (None, 30, 128)           197120

 lstm_13 (LSTM)               (None, 64)                49408

 dense_6 (Dense)              (None, 64)                4160

 dense_7 (Dense)              (None, 32)                2080

 dense_8 (Dense)              (None, 5)                 165

=================================================================
Total params: 1,188,101
Trainable params: 1,188,101
Non-trainable params: 0
_________________________________________________________________
```

The model's optimizer uses the powerful Adam optimizer. Adam algorithm is a first-order optimization algorithm that can replace traditional stochastic gradient descent. It can iteratively update neural network weights based on training data.
After 60 epochs of training, it can be seen from the below figure that the model has converged, which provides the loss value and accuracy during the training.

Analysis of Results



Overall Performance

Loss: 0.099877232
Accuracy: 0.963333309

However, according to the testing experiments on the model, it can be found that the LSTM model performances are not stable and it is sensitive to different random weight initializations. Also, the reason of fluctuations of the performance can be inferred that the model is easy to overfit.

5.2 OVERALL MODEL PERFORMANCE

Model	Accuracy	Prediction Time/s
KNN (Classification)	0.9398148148148148	0.05199146270751953 × 6
Random Forest (Classification)	0.95466666666667	0.04009609222412109 × 6
AdaBoosting (Classification)	0.7946296296296296	0.0488757848739624 × 6
LDA (Classification)	0.9548148148148148	"0.001" × 6
LSTM (Sequential)	0.963333309	0.1167
Transformer (Sequential)	0.9952380657196045	0.076

6. PROJECT AND DEPLOYMENT CONSIDERATIONS

Frames

During model training, 30 frames of sequence length was selected and used to train the model. However, we realised that most camera are only able to capture 30 frames

per second which results in taking at least one second to collect the necessary frames required. For the application of real-time pose classification, taking a second to trigger the rhythmic game input is consider slow relative to the average reaction time for humans at 0.25 seconds.

Therefore, to optimise between response time and accuracy, we have chosen to acquire 6 frames of image for prediction, resulting in an almost real-time pose classification as shown in our demo video.

Why not use keras.js/ OpenCV.js

Initially we planned to do everything on the frontend as speed is a critical part as we are doing a real time processing. However, the latest keras.js doesn't support the transformer model and we have to fall back to use socket connection to a server which provides almost a real time response. For opencv.js, we planned to use it to do video processing but as it is a large library and it takes really long time to process the frames so we decided to use the browser built in camera function to capture the video.