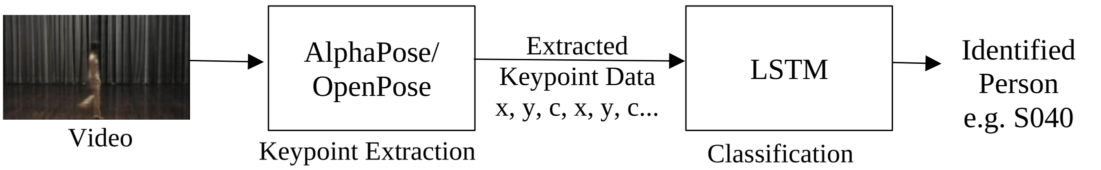
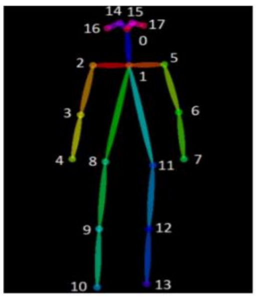
Individual Contribution:

Name	Model
LIM YUAN HER (20A459H)	Using AlphaPose/Openpose and LSTM for Gait Recognition

- **Model Development Log**
  - **Input**
    - Video files in .avi format
    - Data from *GaHu-Video: Parametrization system for human gait recognition* dataset downloadable at: https://md-datasets-cache-zipfiles-prod.s3.eu-west-1.amazonaws.com/gprg4s73v4-2.zip
  - **Output**
    - Person identifier e.g. S040
  - **Architecture**



The input videos are preprocessed using pose estimation models (AlphaPose/OpenPose) to extract keypoint data from each video frame and the sequence data are used as input to a LSTM network for classification modelling.



- 0: Nose X, Y
- 1: Neck X, Y
- 2: Right Shoulder X, Y
- 3: Right Elbow X, Y
- 4: Right Wrist X, Y
- 5: Left Shoulder X, Y
- 6: Left Elbow X, Y
- 7: Left Wrist X, Y
- 8: Right Hip X, Y
- 9: Right Knee X, Y
- 10: Right Ankle X, Y
- 11: Left Hip X, Y
- 12: Left Knee X, Y
- 13: Left Ankle X, Y
- 14: Right Eye X, Y
- 15: Left Eye X, Y
- 16: Right Ear X, Y
- 17: Left Ear X, Y

  - **Data Preprocessing**
    - Removal of the detection confidence score (C) from each keypoint data pair (X, Y, C)
    - Use neck point (point 1) as centre of entire skeleton from which other keypoint data pair take reference from to cater for varying body part lengths of different persons.
    - Scaling based on the torso length to cater for distance from the camera
    - Only keypoint data of first *50* frames from videos (with more than 50 frames extracted using OpenPose/AlphaPose) were included to exclude data from poor quality videos

- **Model Summary**
  - As extracted keypoint data is sequential data, LSTM-based network is used
  - Simple 1-layer, stacked, bidirectional LSTM networks were evaluated:

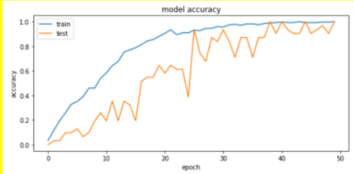
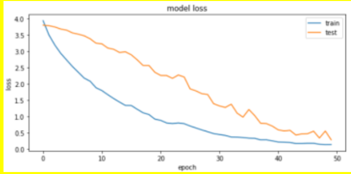
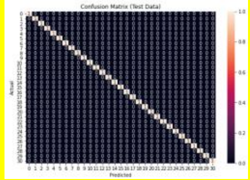
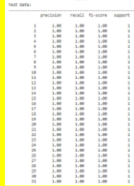
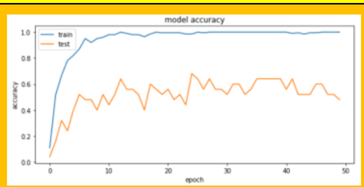
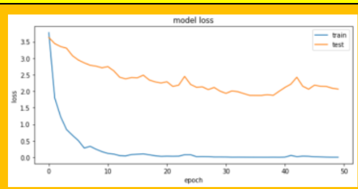
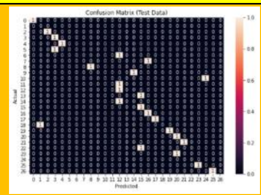
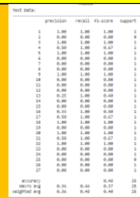
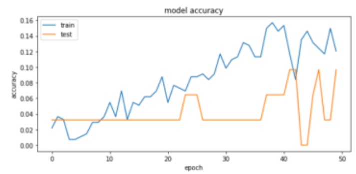
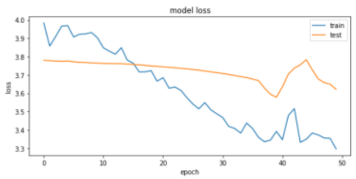
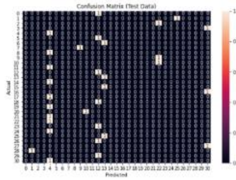
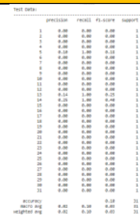
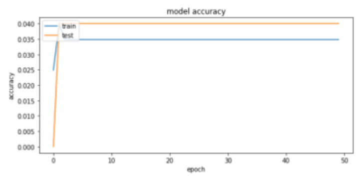
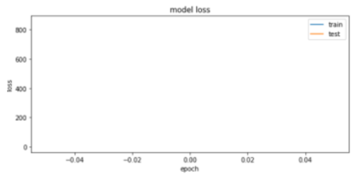
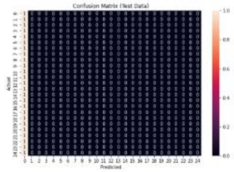
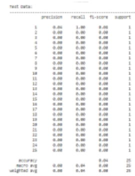
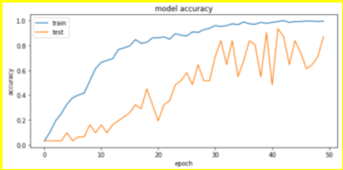
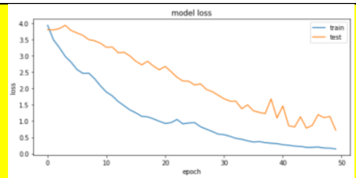
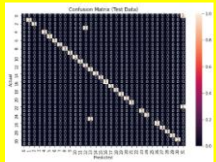
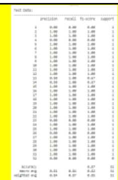
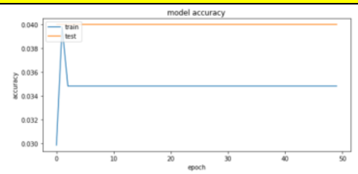
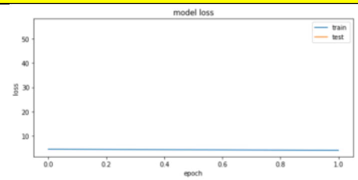
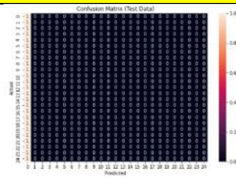
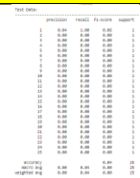
Type	Model Summary	Code

Simple	Model: "sequential_21"
	Layer (type) Output Shape Param #
	lstm_29 (LSTM) (None, 64) 25856
	batch_normalization_29 (Batc (None, 64) 256
	dense_21 (Dense) (None, 44) 2860
	Total params: 28,972
	Trainable params: 28,844
	Non-trainable params: 128
		def createSimpleLSTM(neurons=64):
		# Use Keras to create a Sequential model here with any layers that
		# you'd like.
		#
		model = Sequential()
		model.add(LSTM(neurons, input_shape=(50, 36)))
		model.add(BatchNormalization())
		# One or more repetitions of Dense layers
		model.add(Dense(44, activation='softmax'))
		model.compile(loss=categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
		model.summary()
		return model

Stacked	Layer (type) Output Shape Param #
	lstm_30 (LSTM) (None, 50, 64) 25856
	batch_normalization_30 (Batc (None, 50, 64) 256
	lstm_31 (LSTM) (None, 50, 32) 12416
	batch_normalization_31 (Batc (None, 50, 32) 128
	lstm_32 (LSTM) (None, 16) 3136
	batch_normalization_32 (Batc (None, 16) 64
	dense_22 (Dense) (None, 44) 748
	Total params: 42,604
	Trainable params: 42,380
	Non-trainable params: 224
		def createStackedLSTM(neurons=64):
		# Use Keras to create a Sequential model here with any layers that
		# you'd like.
		#
		model = Sequential()
		layer2_neurons = int(neurons/2)
		layer3_neurons = int(neurons/4)
		#Stacked
		model.add(LSTM(neurons, activation='relu', return_sequences=True, input_shape=(50, 36)))
		model.add(BatchNormalization())
		model.add(LSTM(layer2_neurons, activation='relu', return_sequences=True))
		model.add(BatchNormalization())
		model.add(LSTM(layer3_neurons, activation='relu'))
		model.add(BatchNormalization())
		# One or more repetitions of Dense layers
		model.add(Dense(44, activation='softmax'))
		model.compile(loss=categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
		model.summary()
		return model

BiDirectional	Layer (type) Output Shape Param #
	bidirectional_4 (Bidirection (None, 128) 51712
	batch_normalization_33 (Batc (None, 128) 512
	dense_23 (Dense) (None, 44) 5676
	Total params: 57,900
	Trainable params: 57,644
	Non-trainable params: 256
		def createBiLSTM(neurons=64):
		# Use Keras to create a Sequential model here with any layers that
		# you'd like.
		#
		model = Sequential()
		#Bidirectional
		model.add(Bidirectional(LSTM(neurons, activation='relu'), input_shape=(50, 36)))
		model.add(BatchNormalization())
		# One or more repetitions of Dense layers
		model.add(Dense(44, activation='softmax'))
		model.compile(loss=categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
		model.summary()
		return model

- **Train/Validation/Test Dataset Split**
  - Train/Validation dataset
    - Train – keypoint data extracted from video files in Track A and B
    - Validation – keypoint data extracted from video files in Track C
    - Train/Validation data in *90:10* proportion
  - Test dataset
    - 4 video files were removed from the original dataset and reserved for model testing purposes
- **Hyperparameter**
  - Hyperparameter to be tuned is the number of neurons in each layer e.g. 32, 64, 128 etc.
  - Baseline parameter used: *64* (AlphaPose outputs) and *512* (OpenPose outputs)
- **Epochs**
  - 50 epochs were used for model training.
- **Learning Graphs**
  - Accuracy/loss curves, confusion matrix, classification report for each model run is tabulated as below:

LSTM Network	Pose Estimation	Train/Test Accuracy Curves	Train/Test Loss Curves	Confusion Matrix	Classification Report
Simple	AlphaPose				
	OpenPose				
Stacked	AlphaPose				
	OpenPose				
Bidirectional	AlphaPose				
	OpenPose				

Based on the results above, combination of AlphaPose keypoint data with 1-layer simple or Bi-Directional LSTM network for classification modelling outperformed the rest (highlighted in yellow) whilst using OpenPose keypoint data with a simple 1-layer LSTM network displayed high bias characteristics (highlighted in orange).

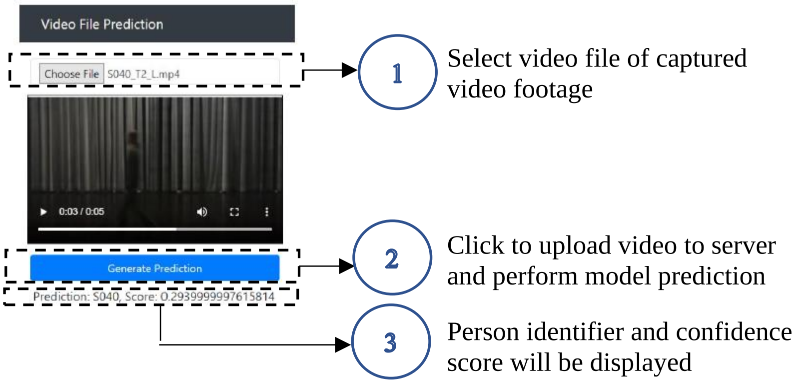
- **Testing Accuracy**

  Based on the results above, using **AlphaPose** keypoint extraction outputs and a **simple 1-layer LSTM** network produced high train and test accuracies above 90%.

- **Possible improvements for next iteration**

  Other possible future enhancements that can be incorporated include:
  - Data augmentation by using a sliding window approach to increase the amount of training data
  - Incorporating detection confidence score of the keypoint extraction output in the training data to check whether it enhances performance
  - Evaluate using other recurrent neural network architectures e.g. GRU (gated recurrent unit) for the classification modelling

- **Model Deployment**
  - The proposed mode involves using a Flask web application packaged as a docker image for deployment to a docker container environment e.g. cloud-based like Amazon ACS, Google CloudRun, Azure ACI etc.
  - The web interface allows the user to select a captured video footage of the persons to be identified and click a "Generate Prediction" button to upload the video for keypoint extraction using Alphapose and perform model prediction using the extracted keypoint data. The person identifier and confidence score will be displayed as shown below:



- **Code**
  - The Jupyter Notebook and source code for the Flask web application is available at the github repository (https://github.com/yuanher/GaitRecognition)
  - The docker image is available from hub.docker.com at yuanher/webapp.