Show code

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Human Pose Classification with MoveNet and TensorFlow Lite

This notebook teaches you how to train a pose classification model using MoveNet and TensorFlow Lite. The result is a new TensorFlow Lite model that accepts the output from the MoveNet model as its input, and outputs a pose classification, such as the name of a yoga pose.

The procedure in this notebook consists of 3 parts:

- Part 1: Preprocess the pose classification training data into a CSV file that specifies the landmarks (body keypoints) detected by the MoveNet model, along with the ground truth pose labels.
- Part 2: Build and train a pose classification model that takes the landmark coordinates from the CSV file as input, and outputs the predicted labels.
- Part 3: Convert the pose classification model to TFLite.

By default, this notebook uses an image dataset with labeled yoga poses, but we've also included a section in Part 1 where you can upload your own image dataset of poses.

 View on TensorFlow.org  Run in Google Colab  View source on GitHub  Download notebook  See TF Hub model

Preparation

In this section, you'll import the necessary libraries and define several functions to preprocess the training images into a CSV file that contains the landmark coordinates and ground truth labels.

Nothing observable happens here, but you can expand the hidden code cells to see the implementation for some of the functions we'll be calling later on.

If you only want to create the CSV file without knowing all the details, just run this section and proceed to Part 1.

```
!pip install -q opencv-python
```

```
import csv
import cv2
import itertools
import numpy as np
import pandas as pd
import os
import sys
import tempfile
import tqdm

from matplotlib import pyplot as plt
from matplotlib.collections import LineCollection

import tensorflow as tf
import tensorflow_hub as hub
from tensorflow import keras

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

Code to run pose estimation using MoveNet

Functions to run pose estimation with MoveNet

You'll download the MoveNet Thunder model from TensorFlow Hub, and reuse some inference and visualization logic from the MoveNet Raspberry Pi (Python) sample app to detect landmarks (ear, nose, wrist etc.) from the input images.

Note: You should use the most accurate pose estimation model (i.e. MoveNet Thunder) to detect the keypoints and use them to train the pose classification model to achieve the best accuracy. When running inference, you can use a pose estimation model of your choice (e.g. either MoveNet Lightning or Thunder).

Show code

```
Cloning into 'examples'...
remote: Enumerating objects: 22843, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 22843 (delta 12), reused 18 (delta 5), pack-reused 22813
Receiving objects: 100% (22843/22843), 42.08 MiB | 26.95 MiB/s, done.
Resolving deltas: 100% (12544/12544), done.
```

Functions to visualize the pose estimation results.

Show code

Code to load the images, detect pose landmarks and save them into a CSV file

Show code

(Optional) Code snippet to try out the Movenet pose estimation logic

You can download an image from the internet, run the pose estimation logic on it and plot the detected landmarks on top of the input image.

Note: This code snippet is also useful for debugging when you encounter an image with bad pose classification accuracy. You can run pose estimation on the image and see if the detected landmarks look correct or not before investigating the pose classification logic.

test_image_url: " https://cdn.pixabay.com/photo/2017/03/03/17/30/yoga-2114512_960_720.jpg "

Show code

Part 1: Preprocess the input images

Because the input for our pose classifier is the *output* landmarks from the MoveNet model, we need to generate our training dataset by running labeled images through MoveNet and then capturing all the landmark data and ground truth labels into a CSV file.

The dataset we've provided for this tutorial is a CG-generated yoga pose dataset. It contains images of multiple CG-generated models doing 5 different yoga poses. The directory is already split into a `train` dataset and a `test` dataset.

So in this section, we'll download the yoga dataset and run it through MoveNet so we can capture all the landmarks into a CSV file... **However, it takes about 15 minutes to feed our yoga dataset to MoveNet and generate this CSV file**. So as an alternative, you can download a pre-existing CSV file for the yoga dataset by setting `is_skip_step_1` parameter below to **True**. That way, you'll skip this step and instead download the same CSV file that will be created in this preprocessing step.

On the other hand, if you want to train the pose classifier with your own image dataset, you need to upload your images and run this preprocessing step (leave `is_skip_step_1` **False**)—follow the instructions below to upload your own pose dataset.

is_skip_step_1: False

Show code

(Optional) Upload your own pose dataset

use_custom_dataset: True

dataset_is_split: False

Show code

If you want to train the pose classifier with your own labeled poses (they can be any poses, not just yoga poses), follow these steps:

1. Set the above `use_custom_dataset` option to **True**.
2. Prepare an archive file (ZIP, TAR, or other) that includes a folder with your images dataset. The folder must include sorted images of your poses as follows.

   If you've already split your dataset into train and test sets, then set `dataset_is_split` to **True**. That is, your images folder must include "train" and "test" directories like this:

```
yoga_poses/
|__ train/
    |__ downdog/
        |______ 00000128.jpg
        |______ ...
|__ test/
    |__ downdog/
        |______ 00000181.jpg
        |______ ...
```

   Or, if your dataset is NOT split yet, then set `dataset_is_split` to **False** and we'll split it up based on a specified split fraction. That is, your uploaded images folder should look like this:

```
yoga_poses/
|__ downdog/
    |______ 00000128.jpg
    |______ 00000181.jpg
    |______ ...
|__ goddess/
    |______ 00000243.jpg
    |______ 00000306.jpg
    |______ ...
```

3. Click the **Files** tab on the left (folder icon) and then click **Upload to session storage** (file icon).
4. Select your archive file and wait until it finishes uploading before you proceed.
5. Edit the following code block to specify the name of your archive file and images directory. (By default, we expect a ZIP file, so you'll need to also modify that part if your archive is another format.)
6. Now run the rest of the notebook.

Be sure you run this cell. It's hiding the `split_into_train_test()` function that's called in the next code block.

Show code

```
if use_custom_dataset:
  # ATTENTION:
  # You must edit these two lines to match your archive and images folder name:
  # !tar -xf YOUR_DATASET_ARCHIVE_NAME.tar
  !unzip -q /content/drive/MyDrive/airost_data/min_sitting_posture.zip
  dataset_in = 'sitting_posture'

  # You can leave the rest alone:
  if not os.path.isdir(dataset_in):
    raise Exception("dataset_in is not a valid directory")
  if dataset_is_split:
    IMAGES_ROOT = dataset_in
  else:
    dataset_out = 'split_' + dataset_in
```

```
split_into_train_test(dataset_in, dataset_out, test_split=0.2)
IMAGES_ROOT = dataset_out
```

```
 replace sitting_posture/crossedleg/20221206_213638_051.jpg? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
 Moved 36 of 180 from class "crossedleg" into test.
 Moved 40 of 200 from class "straight" into test.
 Moved 36 of 180 from class "hunchback" into test.
 Your split dataset is in "split_sitting_posture"
```

Note: If you're using `split_into_train_test()` to split the dataset, it expects all images to be PNG, JPEG, or BMP—it ignores other file types.

Download the yoga dataset

```
if not is_skip_step_1 and not use_custom_dataset:
  !wget -O yoga_poses.zip http://download.tensorflow.org/data/pose_classification/yoga_poses.zip
  !unzip -q yoga_poses.zip -d yoga_cg
  IMAGES_ROOT = "yoga_cg"
```

Preprocess the `TRAIN` dataset

```
if not is_skip_step_1:
  images_in_train_folder = os.path.join(IMAGES_ROOT, 'train')
  images_out_train_folder = 'poses_images_out_train'
  csvs_out_train_path = 'train_data.csv'

  preprocessor = MoveNetPreprocessor(
      images_in_folder=images_in_train_folder,
      images_out_folder=images_out_train_folder,
      csvs_out_path=csvs_out_train_path,
  )

  preprocessor.process(per_pose_class_limit=None)
```

```
Skipped split_sitting_posture/train/straight/20221206_213129_026.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_027.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_028.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_029.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_035.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_036.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_039.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_041.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_042.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_043.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_048.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_049.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/train/straight/20221206_213129_050.jpg. No pose was confidentlly detected.
```

Preprocess the TEST dataset

```
if not is_skip_step_1:
  images_in_test_folder = os.path.join(IMAGES_ROOT, 'test')
  images_out_test_folder = 'poses_images_out_test'
  csvs_out_test_path = 'test_data.csv'

  preprocessor = MoveNetPreprocessor(
      images_in_folder=images_in_test_folder,
      images_out_folder=images_out_test_folder,
      csvs_out_path=csvs_out_test_path,
  )

  preprocessor.process(per_pose_class_limit=None)
```

```
    coordinates = pose_landmarks.flatten().astype(np.str).tolist()
100%|██████████| 36/36 [02:20<00:00,  3.91s/it]
Preprocessing hunchback
100%|██████████| 36/36 [02:18<00:00,  3.85s/it]
Preprocessing straight
100%|██████████| 40/40 [02:35<00:00,  3.88s/it]Skipped split_sitting_posture/test/crossedleg/20221206_213702_078.jpg. No pose was con
Skipped split_sitting_posture/test/crossedleg/20221206_213702_080.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213702_091.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213732_018.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213732_039.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213732_040.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_013.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_014.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_016.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_018.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_022.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_037.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_039.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_041.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213809_044.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213824_007.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213824_014.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213824_015.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/crossedleg/20221206_213824_019.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213349_054.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213349_070.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213349_071.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213349_075.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_002.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_019.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_025.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_031.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_033.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_041.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213423_042.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213606_013.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213606_016.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213606_019.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213606_031.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/hunchback/20221206_213606_033.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_212035.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_212036.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_212633_016.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_213250_004.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_213304_003.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_213304_028.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_213311_007.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221205_213311_008.jpg. No pose was confidentlly detected.
```

```
Skipped split_sitting_posture/test/straight/20221206_213129_008.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221206_213129_038.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221206_213129_044.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221206_213129_045.jpg. No pose was confidentlly detected.
Skipped split_sitting_posture/test/straight/20221206_213129_046.jpg. No pose was confidentlly detected.
```

Part 2: Train a pose classification model that takes the landmark coordinates as input, and output the predicted labels.

You'll build a TensorFlow model that takes the landmark coordinates and predicts the pose class that the person in the input image performs. The model consists of two submodels:

- Submodel 1 calculates a pose embedding (a.k.a feature vector) from the detected landmark coordinates.
- Submodel 2 feeds pose embedding through several `Dense` layer to predict the pose class.

You'll then train the model based on the dataset that were preprocessed in part 1.

(Optional) Download the preprocessed dataset if you didn't run part 1

```
# Download the preprocessed CSV files which are the same as the output of step 1
if is_skip_step_1:
  !wget -O train_data.csv http://download.tensorflow.org/data/pose_classification/yoga_train_data.csv
  !wget -O test_data.csv http://download.tensorflow.org/data/pose_classification/yoga_test_data.csv

  csvs_out_train_path = 'train_data.csv'
  csvs_out_test_path = 'test_data.csv'
  is_skipped_step_1 = True
```

Load the preprocessed CSVs into `TRAIN` and `TEST` datasets.

```
def load_pose_landmarks(csv_path):
  """Loads a CSV created by MoveNetPreprocessor.

  Returns:
    X: Detected landmark coordinates and scores of shape (N, 17 * 3)
    y: Ground truth labels of shape (N, label_count)
    classes: The list of all class names found in the dataset
    dataframe: The CSV loaded as a Pandas dataframe features (X) and ground
      truth labels (y) to use later to train a pose classification model.
  """

  # Load the CSV file
  dataframe = pd.read_csv(csv_path)
  df_to_process = dataframe.copy()

  # Drop the file_name columns as you don't need it during training.
  df_to_process.drop(columns=['file_name'], inplace=True)

  # Extract the list of class names
  classes = df_to_process.pop('class_name').unique()

  # Extract the labels
  y = df_to_process.pop('class_no')

  # Convert the input features and labels into the correct format for training.
  X = df_to_process.astype('float64')
  y = keras.utils.to_categorical(y)

  return X, y, classes, dataframe
```

Load and split the original `TRAIN` dataset into `TRAIN` (85% of the data) and `VALIDATE` (the remaining 15%).

```
# Load the train data
X, y, class_names, _ = load_pose_landmarks(csvs_out_train_path)

# Split training data (X, y) into (X_train, y_train) and (X_val, y_val)
X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size=0.15)
```

```
# Load the test data
X_test, y_test, _, df_test = load_pose_landmarks(csvs_out_test_path)
```

Define functions to convert the pose landmarks to a pose embedding (a.k.a. feature vector) for pose classification

Next, convert the landmark coordinates to a feature vector by:

1. Moving the pose center to the origin.
2. Scaling the pose so that the pose size becomes 1
3. Flattening these coordinates into a feature vector

Then use this feature vector to train a neural-network based pose classifier.

```
def get_center_point(landmarks, left_bodypart, right_bodypart):
  """Calculates the center point of the two given landmarks."""

  left = tf.gather(landmarks, left_bodypart.value, axis=1)
  right = tf.gather(landmarks, right_bodypart.value, axis=1)
  center = left * 0.5 + right * 0.5
  return center


def get_pose_size(landmarks, torso_size_multiplier=2.5):
  """Calculates pose size.

  It is the maximum of two values:
    * Torso size multiplied by `torso_size_multiplier`
    * Maximum distance from pose center to any pose landmark
  """
  # Hips center
  hips_center = get_center_point(landmarks, BodyPart.LEFT_HIP,
                                 BodyPart.RIGHT_HIP)

  # Shoulders center
  shoulders_center = get_center_point(landmarks, BodyPart.LEFT_SHOULDER,
                                      BodyPart.RIGHT_SHOULDER)

  # Torso size as the minimum body size
  torso_size = tf.linalg.norm(shoulders_center - hips_center)

  # Pose center
  pose_center_new = get_center_point(landmarks, BodyPart.LEFT_HIP,
                                     BodyPart.RIGHT_HIP)
  pose_center_new = tf.expand_dims(pose_center_new, axis=1)
  # Broadcast the pose center to the same size as the landmark vector to
  # perform substraction
  pose_center_new = tf.broadcast_to(pose_center_new,
                                    [tf.size(landmarks) // (17*2), 17, 2])

  # Dist to pose center
  d = tf.gather(landmarks - pose_center_new, 0, axis=0,
                name="dist_to_pose_center")
  # Max dist to pose center
  max_dist = tf.reduce_max(tf.linalg.norm(d, axis=0))

  # Normalize scale
  pose_size = tf.maximum(torso_size * torso_size_multiplier, max_dist)

  return pose_size


def normalize_pose_landmarks(landmarks):
  """Normalizes the landmarks translation by moving the pose center to (0,0) and
  scaling it to a constant pose size.
  """
  # Move landmarks so that the pose center becomes (0,0)
  pose_center = get_center_point(landmarks, BodyPart.LEFT_HIP,
                                 BodyPart.RIGHT_HIP)
  pose_center = tf.expand_dims(pose_center, axis=1)
  # Broadcast the pose center to the same size as the landmark vector to perform
  # substraction
  pose_center = tf.broadcast_to(pose_center,
                                [tf.size(landmarks) // (17*2), 17, 2])
```

```
  landmarks = landmarks - pose_center

  # Scale the landmarks to a constant pose size
  pose_size = get_pose_size(landmarks)
  landmarks /= pose_size

  return landmarks


def landmarks_to_embedding(landmarks_and_scores):
  """Converts the input landmarks into a pose embedding."""
  # Reshape the flat input into a matrix with shape=(17, 3)
  reshaped_inputs = keras.layers.Reshape((17, 3))(landmarks_and_scores)

  # Normalize landmarks 2D
  landmarks = normalize_pose_landmarks(reshaped_inputs[:, :, :2])

  # Flatten the normalized landmark coordinates into a vector
  embedding = keras.layers.Flatten()(landmarks)

  return embedding
```

Define a Keras model for pose classification

Our Keras model takes the detected pose landmarks, then calculates the pose embedding and predicts the pose class.

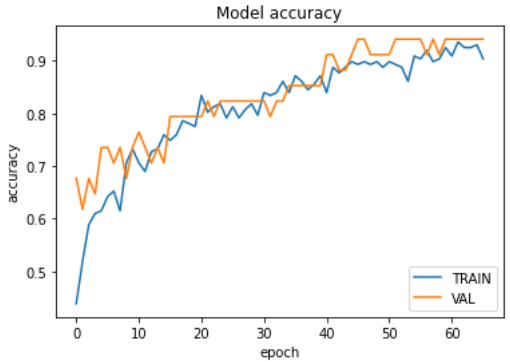
```
# Define the model
inputs = tf.keras.Input(shape=(51))
embedding = landmarks_to_embedding(inputs)

layer = keras.layers.Dense(128, activation=tf.nn.relu6)(embedding)
layer = keras.layers.Dropout(0.5)(layer)
layer = keras.layers.Dense(64, activation=tf.nn.relu6)(layer)
layer = keras.layers.Dropout(0.5)(layer)
outputs = keras.layers.Dense(len(class_names), activation="softmax")(layer)

model = keras.Model(inputs, outputs)
model.summary()
```

```
 flatten (Flatten)              (None, 34)           0           ['tf.math.truediv[0][0]']

 dense (Dense)                  (None, 128)          4480        ['flatten[0][0]']

 dropout (Dropout)              (None, 128)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 64)           8256        ['dropout[0][0]']

 dropout_1 (Dropout)            (None, 64)           0           ['dense_1[0][0]']

 dense_2 (Dense)                (None, 3)            195         ['dropout_1[0][0]']

==================================================================================================
Total params: 12,931
Trainable params: 12,931
Non-trainable params: 0
```

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Add a checkpoint callback to store the checkpoint that has the highest
# validation accuracy.
checkpoint_path = "weights.best.hdf5"
checkpoint = keras.callbacks.ModelCheckpoint(checkpoint_path,
                             monitor='val_accuracy',
                             verbose=1,
                             save_best_only=True,
                             mode='max')
earlystopping = keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                              patience=20)

# Start training
history = model.fit(X_train, y_train,
                    epochs=200,
                    batch_size=16,
                    validation_data=(X_val, y_val),
                    callbacks=[checkpoint, earlystopping])
```

```
 1/12 [=>............................] - ETA: 0s - loss: 0.3886 - accuracy: 0.8750
Epoch 62: val_accuracy did not improve from 0.94118
12/12 [==============================] - 0s 6ms/step - loss: 0.2530 - accuracy: 0.9358 - val_loss: 0.2192 - val_accuracy: 0.9412
Epoch 63/200
 1/12 [=>............................] - ETA: 0s - loss: 0.1994 - accuracy: 0.9375
Epoch 63: val_accuracy did not improve from 0.94118
12/12 [==============================] - 0s 4ms/step - loss: 0.2411 - accuracy: 0.9251 - val_loss: 0.2211 - val_accuracy: 0.9412
Epoch 64/200
 1/12 [=>............................] - ETA: 0s - loss: 0.3396 - accuracy: 0.9375
Epoch 64: val_accuracy did not improve from 0.94118
12/12 [==============================] - 0s 5ms/step - loss: 0.2565 - accuracy: 0.9251 - val_loss: 0.2214 - val_accuracy: 0.9412
Epoch 65/200
 1/12 [=>............................] - ETA: 0s - loss: 0.0956 - accuracy: 1.0000
Epoch 65: val_accuracy did not improve from 0.94118
12/12 [==============================] - 0s 4ms/step - loss: 0.2601 - accuracy: 0.9305 - val_loss: 0.2160 - val_accuracy: 0.9412
Epoch 66/200
 1/12 [=>............................] - ETA: 0s - loss: 0.2827 - accuracy: 0.9375
Epoch 66: val_accuracy did not improve from 0.94118
12/12 [==============================] - 0s 5ms/step - loss: 0.2670 - accuracy: 0.9037 - val_loss: 0.2134 - val_accuracy: 0.9412
```

```
# Visualize the training history to see whether you're overfitting.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['TRAIN', 'VAL'], loc='lower right')
plt.show()
```




```
# Evaluate the model using the TEST dataset
loss, accuracy = model.evaluate(X_test, y_test)
```

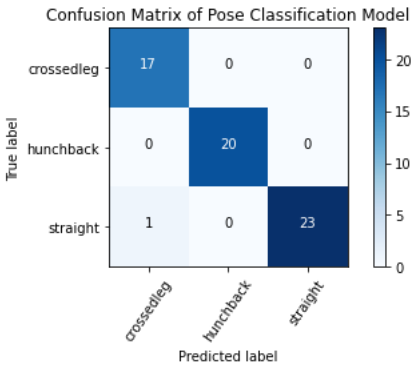
```
2/2 [==============================] - 0s 7ms/step - loss: 0.2449 - accuracy: 0.9836
```

Draw the confusion matrix to better understand the model performance

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
  """Plots the confusion matrix."""
  if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
  else:
    print('Confusion matrix, without normalization')

  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=55)
  plt.yticks(tick_marks, classes)
  fmt = '.2f' if normalize else 'd'
  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

  plt.ylabel('True label')
```

```
  plt.xlabel('Predicted label')
  plt.tight_layout()

# Classify pose in the TEST dataset using the trained model
y_pred = model.predict(X_test)

# Convert the prediction result to class name
y_pred_label = [class_names[i] for i in np.argmax(y_pred, axis=1)]
y_true_label = [class_names[i] for i in np.argmax(y_test, axis=1)]

# Plot the confusion matrix
cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1))
plot_confusion_matrix(cm,
                      class_names,
                      title ='Confusion Matrix of Pose Classification Model')

# Print the classification report
print('\nClassification Report:\n', classification_report(y_true_label,
                                                         y_pred_label))
```

```
2/2 [==============================] - 0s 6ms/step
Confusion matrix, without normalization

Classification Report:
              precision    recall  f1-score   support

   crossedleg       0.94      1.00      0.97        17
    hunchback       1.00      1.00      1.00        20
     straight       1.00      0.96      0.98        24

     accuracy                           0.98        61
    macro avg       0.98      0.99      0.98        61
 weighted avg       0.98      0.98      0.98        61
```



(Optional) Investigate incorrect predictions

You can look at the poses from the `TEST` dataset that were incorrectly predicted to see whether the model accuracy can be improved.

Note: This only works if you have run step 1 because you need the pose image files on your local machine to display them.

```
if is_skip_step_1:
  raise RuntimeError('You must have run step 1 to run this cell.')

# If step 1 was skipped, skip this step.
IMAGE_PER_ROW = 3
MAX_NO_OF_IMAGE_TO_PLOT = 30

# Extract the list of incorrectly predicted poses
false_predict = [id_in_df for id_in_df in range(len(y_test)) \
                 if y_pred_label[id_in_df] != y_true_label[id_in_df]]
if len(false_predict) > MAX_NO_OF_IMAGE_TO_PLOT:
  false_predict = false_predict[:MAX_NO_OF_IMAGE_TO_PLOT]

# Plot the incorrectly predicted images
row_count = len(false_predict) // IMAGE_PER_ROW + 1
fig = plt.figure(figsize=(10 * IMAGE_PER_ROW, 10 * row_count))
for i, id_in_df in enumerate(false_predict):
  ax = fig.add_subplot(row_count, IMAGE_PER_ROW, i + 1)
  image_path = os.path.join(images_out_test_folder,
                            df_test.iloc[id_in_df]['file_name'])
```

```
  image = cv2.imread(image_path)
  plt.title("Predict: %s; Actual: %s"
            % (y_pred_label[id_in_df], y_true_label[id_in_df]))
  plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
```

Part 3: Convert the pose classification model to TensorFlow Lite

You'll convert the Keras pose classification model to the TensorFlow Lite format so that you can deploy it to mobile apps, web browsers and edge devices. When converting the model, you'll apply dynamic range quantization to reduce the pose classification TensorFlow Lite model size by about 4 times with insignificant accuracy loss.

Note: TensorFlow Lite supports multiple quantization schemes. See the documentation if you are interested to learn more.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

print('Model size: %dKB' % (len(tflite_model) / 1024))

with open('pose_classifier.tflite', 'wb') as f:
  f.write(tflite_model)
```

```
Model size: 26KB
```

Then you'll write the label file which contains mapping from the class indexes to the human readable class names.

```
with open('pose_labels.txt', 'w') as f:
  f.write('\n'.join(class_names))
```

As you've applied quantization to reduce the model size, let's evaluate the quantized TFLite model to check whether the accuracy drop is acceptable.

```
def evaluate_model(interpreter, X, y_true):
  """Evaluates the given TFLite model and return its accuracy."""
  input_index = interpreter.get_input_details()[0]["index"]
  output_index = interpreter.get_output_details()[0]["index"]

  # Run predictions on all given poses.
  y_pred = []
  for i in range(len(y_true)):
    # Pre-processing: add batch dimension and convert to float32 to match with
    # the model's input data format.
    test_image = X[i: i + 1].astype('float32')
    interpreter.set_tensor(input_index, test_image)

    # Run inference.
    interpreter.invoke()

    # Post-processing: remove batch dimension and find the class with highest
    # probability.
    output = interpreter.tensor(output_index)
    predicted_label = np.argmax(output()[0])
    y_pred.append(predicted_label)

  # Compare prediction results with ground truth labels to calculate accuracy.
  y_pred = keras.utils.to_categorical(y_pred)
  return accuracy_score(y_true, y_pred)

# Evaluate the accuracy of the converted TFLite model
classifier_interpreter = tf.lite.Interpreter(model_content=tflite_model)
classifier_interpreter.allocate_tensors()
print('Accuracy of TFLite model: %s' %
      evaluate_model(classifier_interpreter, X_test, y_test))
```

```
Accuracy of TFLite model: 0.8360655737704918
```

Now you can download the TFLite model (`pose_classifier.tflite`) and the label file (`pose_labels.txt`) to classify custom poses. See the Android and Python/Raspberry Pi sample app for an end-to-end example of how to use the TFLite pose classification model.

```
!zip pose_classifier.zip pose_labels.txt pose_classifier.tflite
```

```
  adding: pose_labels.txt (stored 0%)
  adding: pose_classifier.tflite (deflated 34%)
```

```
# Download the zip archive if running on Colab.
try:
  from google.colab import files
  files.download('pose_classifier.zip')
except:
  pass
```