Import the data from Edge Impulse. You can obtain the URL from the Dashboard, right-click on the download icon next to 'Spectral features data' and 'Spectral features labels', and click **Copy link location**.

```
import numpy as np
import requests

API_KEY = 'ei_854a88bca0a2854887fb9b0cbe296ca75a5c4a4ceb4c7f4d227381a9b119de6a'

X = (requests.get('https://studio.edgeimpulse.com/v1/api/20675/training/5/x', he
Y = (requests.get('https://studio.edgeimpulse.com/v1/api/20675/training/5/y', he
```

Store the data in a temporary file, and load it back through Numpy.

```
with open('x_train.npy', 'wb') as file:
    file.write(X)
with open('y_train.npy', 'wb') as file:
    file.write(Y)
X = np.load('x_train.npy')
Y = np.load('y_train.npy')[:,0]
```

Define our labels and split the data up in a test and training set:

```
import sys, os, random
import tensorflow as tf
from sklearn.model_selection import train_test_split

import logging
tf.get_logger().setLevel(logging.ERROR)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Set random seeds for repeatable results
RANDOM_SEED = 3
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)

classes_values = [ "_noise", "_unknown", "backward", "follow", "forward", "four"
classes = len(classes_values)

Y = tf.keras.utils.to_categorical(Y - 1, classes)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_

input_length = X_train[0].shape[0]

train_dataset = tf.data.Dataset.from_tensor_slices((X_train, Y_train))
validation_dataset = tf.data.Dataset.from_tensor_slices((X_test, Y_test))
```

✓ 0s completed at 12:39 PM

Train the model:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout, Conv1D, Conv2D,
from tensorflow.keras.optimizers import Adam

# model architecture
model = Sequential()
model.add(Reshape((int(input_length / 13), 13), input_shape=(input_length, )))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Dropout(0.25))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='relu', padding='same'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(classes, activation='softmax', name='y_pred'))

# this controls the learning rate
opt = Adam(lr=0.005, beta_1=0.9, beta_2=0.999)
# this controls the batch size, or you can manipulate the tf.data.Dataset object
BATCH_SIZE = 32
train_dataset, validation_dataset = set_batch_size(BATCH_SIZE, train_dataset, va

# train the neural network
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy
model.fit(train_dataset, epochs=200, validation_data=validation_dataset, verbose
```

```
    Epoch 1/200
    540/540 - 6s - loss: 2.2928 - accuracy: 0.2527 - val_loss: 1.7733 - val_acc
    Epoch 2/200
    540/540 - 4s - loss: 1.6485 - accuracy: 0.4643 - val_loss: 1.3441 - val_acc
    Epoch 3/200
    540/540 - 4s - loss: 1.4323 - accuracy: 0.5517 - val_loss: 1.1959 - val_acc
    540/540 - 4s - loss: 1.1179 - accuracy: 0.6678 - val_loss: 0.9952 - val_acc
    Epoch 11/200
    540/540 - 4s - loss: 1.1091 - accuracy: 0.6679 - val_loss: 1.0269 - val_acc
    Epoch 12/200
    540/540 - 4s - loss: 1.1143 - accuracy: 0.6679 - val_loss: 1.0113 - val_acc
    Epoch 13/200
    540/540 - 4s - loss: 1.0906 - accuracy: 0.6790 - val_loss: 1.0166 - val_acc
    Epoch 14/200
    540/540 - 4s - loss: 1.0843 - accuracy: 0.6776 - val_loss: 1.0429 - val_acc
    Epoch 15/200
    540/540 - 4s - loss: 1.0613 - accuracy: 0.6852 - val_loss: 1.0064 - val_acc
    Epoch 16/200
    540/540 - 4s - loss: 1.0464 - accuracy: 0.6934 - val_loss: 0.9891 - val_acc
    Epoch 17/200
    540/540 - 4s - loss: 1.0460 - accuracy: 0.6909 - val_loss: 1.0164 - val_acc
    Epoch 18/200
    540/540 - 4s - loss: 1.0649 - accuracy: 0.6888 - val_loss: 1.0132 - val_acc
    Epoch 19/200
    540/540 - 4s - loss: 1.0583 - accuracy: 0.6847 - val_loss: 1.0820 - val_acc
    Epoch 20/200
    540/540 - 4s - loss: 1.0609 - accuracy: 0.6976 - val_loss: 1.0094 - val_acc
    Epoch 21/200
    540/540 - 4s - loss: 1.0404 - accuracy: 0.6928 - val_loss: 1.0162 - val_acc
    Epoch 22/200
    540/540 - 4s - loss: 1.0097 - accuracy: 0.7045 - val_loss: 0.9721 - val_acc
    Epoch 23/200
    540/540 - 4s - loss: 1.0465 - accuracy: 0.6938 - val_loss: 1.0714 - val_acc
    Epoch 24/200
    540/540 - 4s - loss: 1.0347 - accuracy: 0.6998 - val_loss: 0.9989 - val_acc
    Epoch 25/200
    540/540 - 4s - loss: 1.0106 - accuracy: 0.7053 - val_loss: 1.0194 - val_acc
    Epoch 26/200
    540/540 - 4s - loss: 1.0336 - accuracy: 0.7036 - val_loss: 0.9827 - val_acc
    Epoch 27/200
    540/540 - 4s - loss: 1.0137 - accuracy: 0.7061 - val_loss: 1.0050 - val_acc
    Epoch 28/200
    540/540 - 4s - loss: 1.0376 - accuracy: 0.6987 - val_loss: 1.0087 - val_acc
    Epoch 29/200
    540/540 - 4s - loss: 1.0335 - accuracy: 0.7045 - val_loss: 1.0347 - val_acc
```

```
# Save the model to disk
model.save('saved_model')
```

```
model.summary()
```

```
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 6, 32)             3104      
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 3, 32)             0         
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 3, 32)             3104      
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 2, 32)             0         
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 2, 32)             3104      
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 1, 32)             0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 32)             0         
_________________________________________________________________
flatten (Flatten)            (None, 32)                0         
_________________________________________________________________
y_pred (Dense)               (None, 18)                594       
=================================================================
Total params: 17,394
Trainable params: 17,394
Non-trainable params: 0
_________________________________________________________________
```