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", "forward", "go", "left", "n
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))
```

✓ 9m 48s completed at 6:48 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
    450/450 - 4s - loss: 2.2549 - accuracy: 0.2302 - val_loss: 1.5075 - val_acc
    Epoch 2/200
    450/450 - 3s - loss: 1.4990 - accuracy: 0.4990 - val_loss: 1.2293 - val_acc
    Epoch 3/200
    450/450 - 3s - loss: 1.3216 - accuracy: 0.5661 - val_loss: 1.1017 - val_acc
```

```
450/450 - 3s - loss: 0.9057 - accuracy: 0.7055 - val_loss: 0.9422 - val_acc
Epoch 11/200
450/450 - 3s - loss: 0.9541 - accuracy: 0.7135 - val_loss: 0.9134 - val_acc
Epoch 12/200
450/450 - 3s - loss: 0.9705 - accuracy: 0.7083 - val_loss: 0.9008 - val_acc
Epoch 13/200
450/450 - 3s - loss: 0.9269 - accuracy: 0.7171 - val_loss: 0.9118 - val_acc
Epoch 14/200
450/450 - 3s - loss: 0.9334 - accuracy: 0.7156 - val_loss: 0.9085 - val_acc
Epoch 15/200
450/450 - 3s - loss: 0.9195 - accuracy: 0.7226 - val_loss: 0.9074 - val_acc
Epoch 16/200
450/450 - 3s - loss: 0.9135 - accuracy: 0.7245 - val_loss: 0.9060 - val_acc
Epoch 17/200
450/450 - 3s - loss: 0.9213 - accuracy: 0.7269 - val_loss: 0.9211 - val_acc
Epoch 18/200
450/450 - 3s - loss: 0.9182 - accuracy: 0.7272 - val_loss: 0.9162 - val_acc
Epoch 19/200
450/450 - 3s - loss: 0.8904 - accuracy: 0.7349 - val_loss: 0.9419 - val_acc
Epoch 20/200
450/450 - 3s - loss: 0.8962 - accuracy: 0.7335 - val_loss: 0.8895 - val_acc
Epoch 21/200
450/450 - 3s - loss: 0.8837 - accuracy: 0.7396 - val_loss: 0.8855 - val_acc
Epoch 22/200
450/450 - 3s - loss: 0.8483 - accuracy: 0.7438 - val_loss: 0.9274 - val_acc
Epoch 23/200
450/450 - 3s - loss: 0.8784 - accuracy: 0.7391 - val_loss: 0.9264 - val_acc
Epoch 24/200
450/450 - 3s - loss: 0.8479 - accuracy: 0.7462 - val_loss: 0.8807 - val_acc
Epoch 25/200
450/450 - 3s - loss: 0.8604 - accuracy: 0.7451 - val_loss: 0.8976 - val_acc
Epoch 26/200
450/450 - 3s - loss: 0.8489 - accuracy: 0.7440 - val_loss: 0.8740 - val_acc
Epoch 27/200
450/450 - 3s - loss: 0.8723 - accuracy: 0.7407 - val_loss: 0.8926 - val_acc
Epoch 28/200
450/450 - 3s - loss: 0.8590 - accuracy: 0.7464 - val_loss: 0.9269 - val_acc
Epoch 29/200
450/450 - 3s - loss: 0.8421 - accuracy: 0.7502 - val_loss: 0.8896 - val_acc
```

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