CNN_training

December 17, 2020

```
[1]: %load_ext autoreload
%autoreload 2

# general imports
import numpy as np
import pickle, time, datetime, itertools, multiprocessing
import matplotlib.pyplot as plt
```

Tensorflow and sklearn imports

```
[10]: import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv1D, MaxPooling1D,␣
 ↪Flatten, MaxPool1D, Conv2D, BatchNormalization, Activation, concatenate
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.constraints import MaxNorm

from kerastuner.tuners import RandomSearch, hyperband
from kerastuner.engine.hyperparameters import HyperParameters

from tqdm.keras import TqdmCallback

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Custon functions import

```
[11]: from ipynb.fs.full.formata_dados import *
from ipynb.fs.full.processa_dados import *
```

0.1 Importing the dataset and creating the data input and output for the network

All the data was treated and imported using the data_cleaning notebook

```
[12]: dataset_path = 'Dataset\\dados_pickle\\'  # folder for the already threated␣
 ↪dataset
# If you are on linux or mac os, use / instead of \\
```

```
with open(dataset_path + "dados_semruido_ok.pkl", "rb") as dados_ok_semruidos:
        dados_sem_ruidos = pickle.load(dados_ok_semruidos)[0]
with open(dataset_path + "dados_semruido_agua_ok.pkl", "rb") as dados_agua_ok:
        dados_semruido_agua_ok= pickle.load(dados_agua_ok)[0]
with open(dataset_path + "dados_semruido_aspartame_ok.pkl", "rb") as␣
 ↪dados_aspartame_ok:
        dados_semruido_aspartame_ok = pickle.load(dados_aspartame_ok)[0]
with open(dataset_path + "dados_semruido_sucralose_ok.pkl", "rb") as␣
 ↪dados_sucralose_ok:
        dados_semruido_sucralose_ok = pickle.load(dados_sucralose_ok)[0]
with open(dataset_path + "dados_semruido_acucar_ok.pkl", "rb") as␣
 ↪dados_acucar_ok:
        dados_semruido_acucar_ok = pickle.load(dados_acucar_ok)[0]
```

[13]:
```
X = dados_sem_ruidos[:,:-4]
y = dados_sem_ruidos[:,-4:]
X_agua = dados_semruido_agua_ok[:,:-4]
y_agua = dados_semruido_agua_ok[:,-4:]
X_acucar = dados_semruido_acucar_ok[:,:-4]
y_acucar = dados_semruido_acucar_ok[:,-4:]
X_aspartame = dados_semruido_aspartame_ok[:,:-4]
y_aspartame = dados_semruido_aspartame_ok[:,-4:]
X_sucralose = dados_semruido_sucralose_ok[:,:-4]
y_sucralose = dados_semruido_sucralose_ok[:,-4:]
```

Options parameters for data multiplication

[99]:
```
window_time = 5        # the size of each data sample in seconds - array size =␣
 ↪window_time * 512
window_slide = 0.001    # the stride in seconds
t_inicial = 0.0        # start value for the data in seconds (time before that␣
 ↪will be ignored)
delta_time = 6         # the amount of data that will be used (time after␣
 ↪delta_time + t_inicial will be ignored)
```

[100]:
```
liquidos = ['agua   ', 'acucar  ', 'aspartame', 'sucralose']

X_agua_train, X_agua_test, y_agua_train, y_agua_test = train_test_split(X_agua,␣
 ↪y_agua, test_size=0.3, random_state=42)
X_acucar_train, X_acucar_test, y_acucar_train, y_agua_test =␣
 ↪train_test_split(X_acucar, y_acucar, test_size=0.3, random_state=42)
X_aspartame_train, X_aspartame_test, y_aspartame_train, y_agua_test =␣
 ↪train_test_split(X_aspartame, y_aspartame, test_size=0.3, random_state=42)
X_sucralose_train, X_sucralose_test, y_sucralose_train, y_sucralose_test =␣
 ↪train_test_split(X_sucralose, y_sucralose, test_size=0.3, random_state=42)
```

```
X_agua_train, y_agua_train = multiplica_dados(X_agua_train,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_agua)
X_acucar_train, y_acucar_train = multiplica_dados(X_acucar_train,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_acucar)
X_aspartame_train, y_aspartame_train = multiplica_dados(X_aspartame_train,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_aspartame)
X_sucralose_train, y_sucralose_train = multiplica_dados(X_sucralose_train,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_sucralose)

X_agua_test, y_agua_test = multiplica_dados(X_agua_test,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_agua)
X_acucar_test, y_acucar_test = multiplica_dados(X_acucar_test,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_acucar)
X_aspartame_test, y_aspartame_test = multiplica_dados(X_aspartame_test,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_aspartame)
X_sucralose_test, y_sucralose_test = multiplica_dados(X_sucralose_test,t_inicial ,t_inicial+delta_time ,512 ,window_time, window_slide, y_sucralose)

y_agua_train = y_agua_train[:,1]
y_acucar_train = y_acucar_train[:,1]
y_aspartame_train = y_aspartame_train[:,1]
y_sucralose_train = y_sucralose_train[:,1]
y_agua_test = y_agua_test[:,1]
y_acucar_test = y_acucar_test[:,1]
y_aspartame_test = y_aspartame_test[:,1]
y_sucralose_test = y_sucralose_test[:,1]

X_train = np.concatenate((X_agua_train, X_acucar_train, X_aspartame_train, X_sucralose_train))
X_test = np.concatenate((X_agua_test, X_acucar_test, X_aspartame_test, X_sucralose_test))
y_train = np.concatenate((y_agua_train, y_acucar_train, y_aspartame_train, y_sucralose_train))
y_test = np.concatenate((y_agua_test, y_acucar_test, y_aspartame_test, y_sucralose_test))

y_train = tf.keras.utils.to_categorical(y_train, num_classes=None, dtype='float32')
y_test = tf.keras.utils.to_categorical(y_test, num_classes=None, dtype='float32')

X_train_len = X_train.shape[0]
```

```
X_all = tf.keras.utils.normalize(np.concatenate((X_train, X_test)), axis=-1,␣
 ↪order=2)
X_train = X_all[:X_train_len,:]
X_test = X_all[X_train_len:,:]

X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
y_train = np.reshape(y_train, (y_train.shape[0], 4))
y_test = np.reshape(y_test, (y_test.shape[0], 4))
```

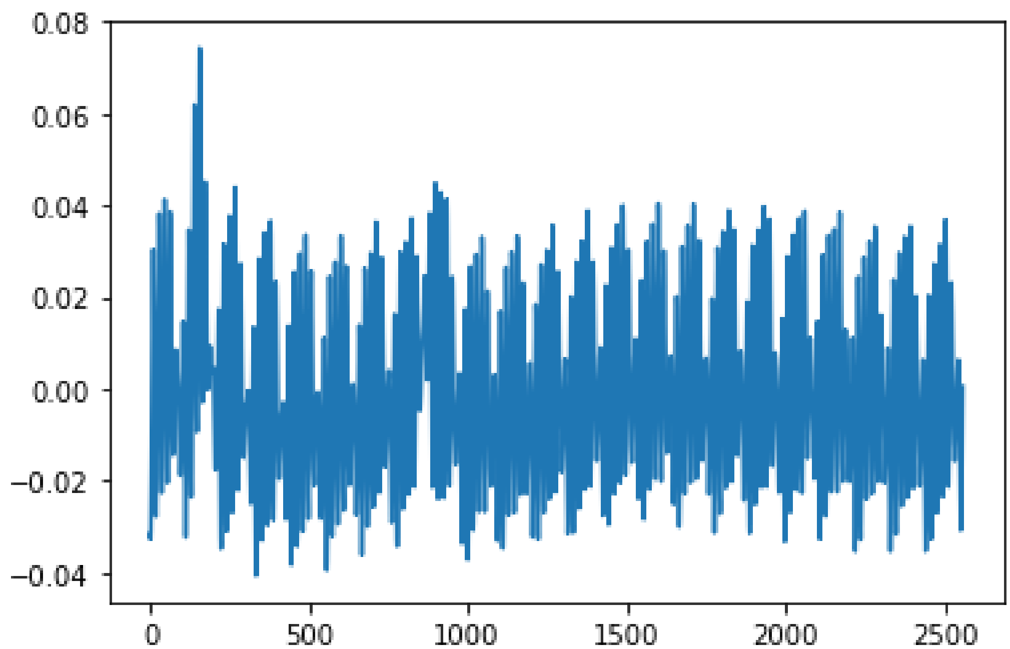
Let's check if data was imported and is in the correct format:

[32]:
```
print(f'Input format: X_train:{X_train.shape} \t y_train:{y_train.shape} \t␣
 ↪X_test:{X_test.shape} \t y_test:{y_test.shape}')
```

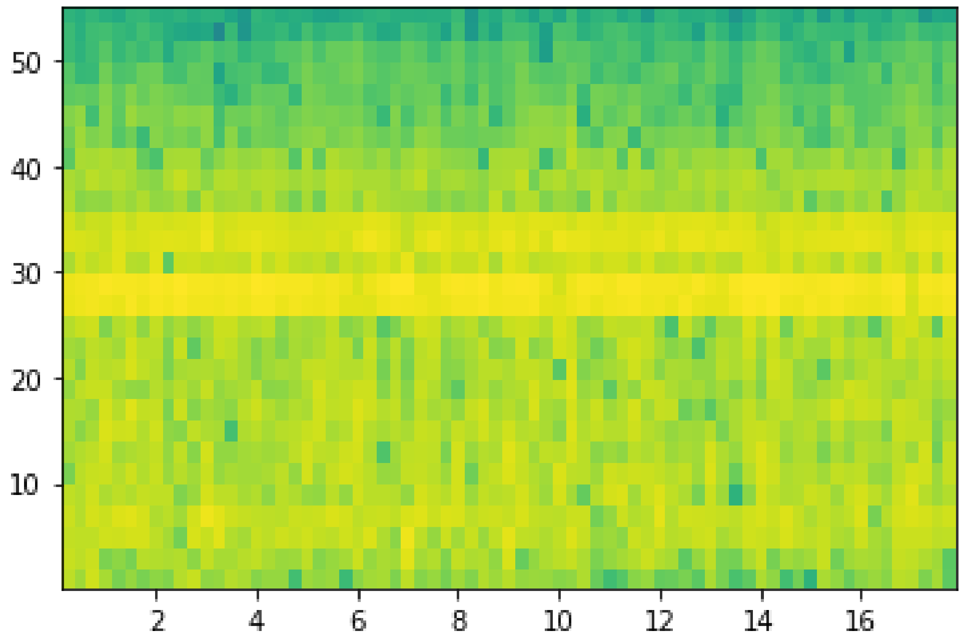
```
Input format: X_train:(2760, 2560, 1)    y_train:(2760, 4)       X_test:(1320,
2560, 1)          y_test:(1320, 4)
```

[33]:
```
plt.plot(X_train[0])
```

[33]:
```
[<matplotlib.lines.Line2D at 0x1790503d608>]
```



[103]:
```
plt.specgram(X_acucar[5], Fs=512)
plt.axis(ymin=0.1, ymax=55)
plt.show()
```



[19]:
```
y_train[0]
```

[19]: array([1., 0., 0., 0.], dtype=float32)

0.2 Now we create the Neural network and train it

Creating a NN with keras fuctional api

[80]:
```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=150)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max',␣
 ↪verbose=2, save_best_only=True)

Inputs = tf.keras.Input(shape=(X_train.shape[1],1))

cv1 = Dropout(0.2)(Inputs)
cv1 = Conv1D(filters=500, kernel_size=1400, activation='relu',␣
 ↪kernel_constraint=MaxNorm(1.), strides=5)(cv1)
cv1 = MaxPooling1D(pool_size=10)(cv1)
cv1 = Flatten()(cv1)
cv1 = Dropout(0.4)(cv1)
cv1 = Dense(200, activation='relu')(cv1)

cv2 = Dropout(0.4)(Inputs)
cv2 = Conv1D(filters=500, kernel_size=150, activation='relu',␣
 ↪kernel_constraint=MaxNorm(1.), strides=5)(cv2)
cv2 = MaxPooling1D(pool_size=10)(cv2)
```

```
cv2 = Flatten()(cv2)
cv2 = Dropout(0.4)(cv2)
cv2 = Dense(200, activation='relu')(cv2)

cv3 = Dropout(0.2)(Inputs)
cv3 = Conv1D(filters=500, kernel_size=500, activation='relu',␣
↪kernel_constraint=MaxNorm(1.), strides=5)(cv3)
cv3 = MaxPooling1D(pool_size=10)(cv3)
cv3 = Flatten()(cv3)
cv3 = Dropout(0.4)(cv3)
cv3 = Dense(200, activation='relu')(cv3)

merge = concatenate([cv1, cv2, cv3])

x = Dropout(0.2)(merge)
x = Dense(400, activation='relu')(x)
x = Dropout(0.2)(x)
Oututs = Dense(4, activation='softmax')(x)


model = tf.keras.Model(inputs=Inputs, outputs=Oututs)
model.compile(
    loss='categorical_crossentropy', #losses.
↪CategoricalCrossentropy(from_logits=True),
    optimizer='adam',
    metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=100, batch_size=32, verbose=0, shuffle=True,␣
↪callbacks=[TqdmCallback(verbose=1), es, mc])
```

```
HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', layout=Layout(width=

HBox(children=(HTML(value=''), FloatProgress(value=1.0, bar_style='info', layout=Layout(width=


Epoch 00001: val_accuracy improved from -inf to 0.31136, saving model to
best_model.h5

Epoch 00002: val_accuracy improved from 0.31136 to 0.39545, saving model to
best_model.h5

Epoch 00003: val_accuracy improved from 0.39545 to 0.41045, saving model to
best_model.h5

Epoch 00004: val_accuracy improved from 0.41045 to 0.41773, saving model to
best_model.h5
```

```
Epoch 00005: val_accuracy improved from 0.41773 to 0.44227, saving model to
best_model.h5

Epoch 00006: val_accuracy did not improve from 0.44227

Epoch 00007: val_accuracy did not improve from 0.44227

Epoch 00008: val_accuracy did not improve from 0.44227

Epoch 00009: val_accuracy did not improve from 0.44227

Epoch 00010: val_accuracy did not improve from 0.44227

Epoch 00011: val_accuracy did not improve from 0.44227

Epoch 00012: val_accuracy did not improve from 0.44227

Epoch 00013: val_accuracy improved from 0.44227 to 0.44455, saving model to
best_model.h5

Epoch 00014: val_accuracy did not improve from 0.44455

Epoch 00015: val_accuracy did not improve from 0.44455

Epoch 00016: val_accuracy did not improve from 0.44455

Epoch 00017: val_accuracy did not improve from 0.44455

Epoch 00018: val_accuracy did not improve from 0.44455

Epoch 00019: val_accuracy did not improve from 0.44455

Epoch 00020: val_accuracy did not improve from 0.44455

Epoch 00021: val_accuracy did not improve from 0.44455

Epoch 00022: val_accuracy did not improve from 0.44455

Epoch 00023: val_accuracy did not improve from 0.44455

Epoch 00024: val_accuracy did not improve from 0.44455

Epoch 00025: val_accuracy did not improve from 0.44455

Epoch 00026: val_accuracy did not improve from 0.44455

Epoch 00027: val_accuracy did not improve from 0.44455
```

```
Epoch 00028: val_accuracy did not improve from 0.44455

Epoch 00029: val_accuracy did not improve from 0.44455

Epoch 00030: val_accuracy did not improve from 0.44455

Epoch 00031: val_accuracy did not improve from 0.44455

Epoch 00032: val_accuracy did not improve from 0.44455

Epoch 00033: val_accuracy did not improve from 0.44455

Epoch 00034: val_accuracy did not improve from 0.44455

Epoch 00035: val_accuracy did not improve from 0.44455

Epoch 00036: val_accuracy did not improve from 0.44455

Epoch 00037: val_accuracy did not improve from 0.44455

Epoch 00038: val_accuracy did not improve from 0.44455

Epoch 00039: val_accuracy did not improve from 0.44455

Epoch 00040: val_accuracy did not improve from 0.44455

Epoch 00041: val_accuracy did not improve from 0.44455

Epoch 00042: val_accuracy did not improve from 0.44455

Epoch 00043: val_accuracy did not improve from 0.44455

Epoch 00044: val_accuracy did not improve from 0.44455

Epoch 00045: val_accuracy did not improve from 0.44455

Epoch 00046: val_accuracy did not improve from 0.44455

Epoch 00047: val_accuracy did not improve from 0.44455

Epoch 00048: val_accuracy did not improve from 0.44455

Epoch 00049: val_accuracy did not improve from 0.44455

Epoch 00050: val_accuracy did not improve from 0.44455

Epoch 00051: val_accuracy did not improve from 0.44455
```

```
Epoch 00052: val_accuracy did not improve from 0.44455

Epoch 00053: val_accuracy did not improve from 0.44455

Epoch 00054: val_accuracy did not improve from 0.44455

Epoch 00055: val_accuracy improved from 0.44455 to 0.45136, saving model to
best_model.h5

Epoch 00056: val_accuracy did not improve from 0.45136

Epoch 00057: val_accuracy did not improve from 0.45136

Epoch 00058: val_accuracy did not improve from 0.45136

Epoch 00059: val_accuracy did not improve from 0.45136

Epoch 00060: val_accuracy did not improve from 0.45136

Epoch 00061: val_accuracy did not improve from 0.45136

Epoch 00062: val_accuracy did not improve from 0.45136

Epoch 00063: val_accuracy did not improve from 0.45136

Epoch 00064: val_accuracy did not improve from 0.45136

Epoch 00065: val_accuracy did not improve from 0.45136

Epoch 00066: val_accuracy did not improve from 0.45136

Epoch 00067: val_accuracy did not improve from 0.45136

Epoch 00068: val_accuracy did not improve from 0.45136

Epoch 00069: val_accuracy did not improve from 0.45136

Epoch 00070: val_accuracy did not improve from 0.45136

Epoch 00071: val_accuracy did not improve from 0.45136

Epoch 00072: val_accuracy did not improve from 0.45136

Epoch 00073: val_accuracy did not improve from 0.45136

Epoch 00074: val_accuracy did not improve from 0.45136
```

```
Epoch 00075: val_accuracy did not improve from 0.45136

Epoch 00076: val_accuracy did not improve from 0.45136

Epoch 00077: val_accuracy did not improve from 0.45136

Epoch 00078: val_accuracy did not improve from 0.45136

Epoch 00079: val_accuracy did not improve from 0.45136

Epoch 00080: val_accuracy did not improve from 0.45136

Epoch 00081: val_accuracy did not improve from 0.45136

Epoch 00082: val_accuracy improved from 0.45136 to 0.47045, saving model to
best_model.h5

Epoch 00083: val_accuracy did not improve from 0.47045

Epoch 00084: val_accuracy did not improve from 0.47045

Epoch 00085: val_accuracy did not improve from 0.47045

Epoch 00086: val_accuracy did not improve from 0.47045

Epoch 00087: val_accuracy did not improve from 0.47045

Epoch 00088: val_accuracy did not improve from 0.47045

Epoch 00089: val_accuracy did not improve from 0.47045

Epoch 00090: val_accuracy did not improve from 0.47045

Epoch 00091: val_accuracy did not improve from 0.47045

Epoch 00092: val_accuracy did not improve from 0.47045

Epoch 00093: val_accuracy did not improve from 0.47045

Epoch 00094: val_accuracy did not improve from 0.47045

Epoch 00095: val_accuracy improved from 0.47045 to 0.48955, saving model to
best_model.h5

Epoch 00096: val_accuracy did not improve from 0.48955

Epoch 00097: val_accuracy did not improve from 0.48955
```

```
Epoch 00098: val_accuracy did not improve from 0.48955

Epoch 00099: val_accuracy did not improve from 0.48955

Epoch 00100: val_accuracy did not improve from 0.48955
```

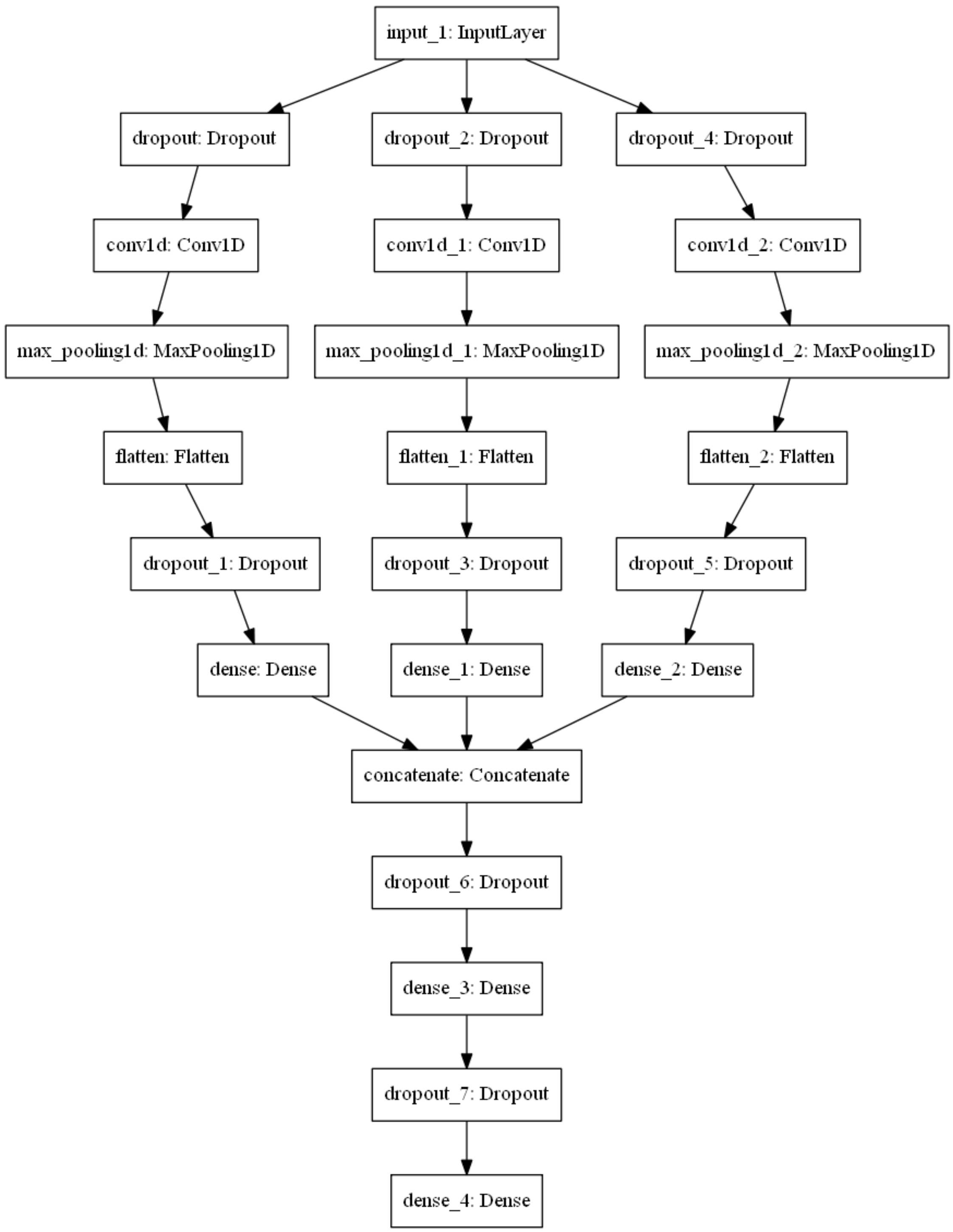
Let's see the best score

[61]:
```
model_best = tf.keras.models.load_model('best_model.h5')
```

This is the NN topology

[62]:
```
tf.keras.utils.plot_model(
    model_best, to_file='model.png', show_shapes=False,
    show_layer_names=True, rankdir='TB', expand_nested=False, dpi=96
)
```

[62]:



Importing and printing our best result

[84]:
```
model_best = tf.keras.models.load_model('Tensorflow_Models\\best_model_601.h5')
```

WARNING:tensorflow:Sequential models without an `input_shape` passed to the

first layer cannot reload their optimizer state. As a result, your model
isstarting with a freshly initialized optimizer.

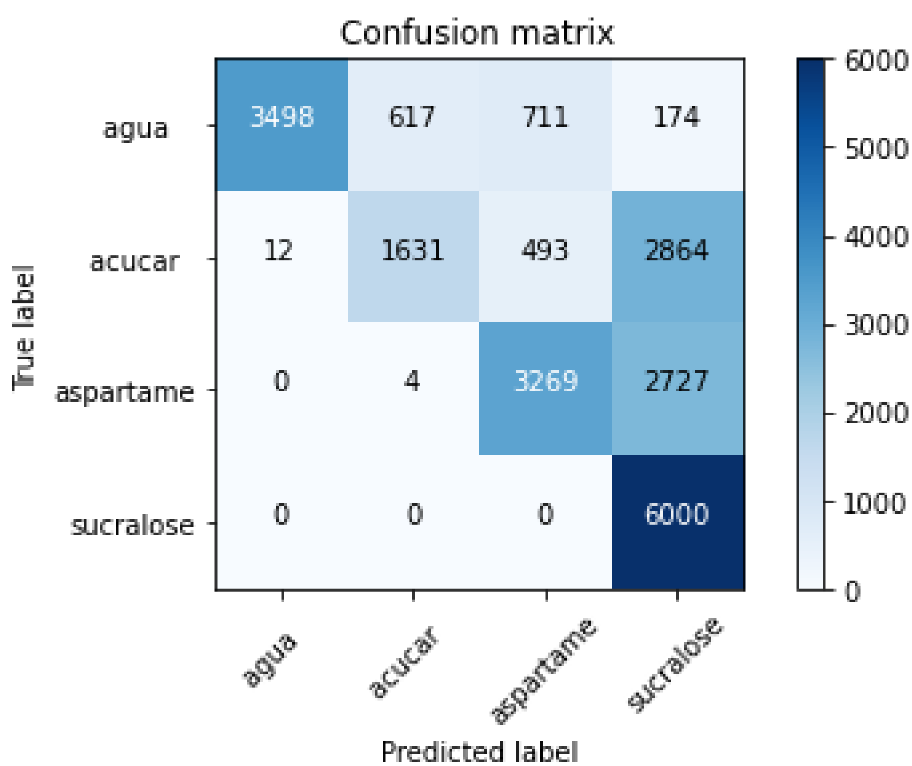
[101]:
```
model_best.evaluate(X_test, y_test)
```

```
22000/22000 [==============================] - ETA: 27s - loss: 0.5261 -
accuracy: 1.000 - ETA: 7s - loss: 0.9620 - accuracy: 0.187 - ETA: 5s - loss:
0.9569 - accuracy: 0.17 - ETA: 5s - loss: 0.9327 - accuracy: 0.18 - ETA: 5s -
loss: 0.8369 - accuracy: 0.32 - ETA: 4s - loss: 0.8368 - accuracy: 0.38 - ETA:
4s - loss: 0.8574 - accuracy: 0.38 - ETA: 4s - loss: 1.1516 - accuracy: 0.32 -
ETA: 4s - loss: 1.2706 - accuracy: 0.35 - ETA: 4s - loss: 1.1083 - accuracy:
0.43 - ETA: 3s - loss: 0.9837 - accuracy: 0.50 - ETA: 3s - loss: 0.8939 -
accuracy: 0.54 - ETA: 3s - loss: 0.8207 - accuracy: 0.58 - ETA: 3s - loss:
0.7554 - accuracy: 0.62 - ETA: 3s - loss: 0.7027 - accuracy: 0.65 - ETA: 3s -
loss: 0.6588 - accuracy: 0.67 - ETA: 3s - loss: 0.6245 - accuracy: 0.69 - ETA:
3s - loss: 0.6118 - accuracy: 0.70 - ETA: 3s - loss: 1.3512 - accuracy: 0.67 -
ETA: 3s - loss: 2.4921 - accuracy: 0.63 - ETA: 3s - loss: 3.5624 - accuracy:
0.60 - ETA: 2s - loss: 4.7915 - accuracy: 0.57 - ETA: 2s - loss: 6.4099 -
accuracy: 0.54 - ETA: 2s - loss: 7.8759 - accuracy: 0.52 - ETA: 2s - loss:
9.1542 - accuracy: 0.50 - ETA: 2s - loss: 8.8766 - accuracy: 0.52 - ETA: 2s -
loss: 8.5395 - accuracy: 0.53 - ETA: 2s - loss: 8.2271 - accuracy: 0.55 - ETA:
2s - loss: 7.9605 - accuracy: 0.55 - ETA: 2s - loss: 7.6997 - accuracy: 0.56 -
ETA: 2s - loss: 7.4536 - accuracy: 0.58 - ETA: 2s - loss: 7.4017 - accuracy:
0.56 - ETA: 2s - loss: 7.8775 - accuracy: 0.54 - ETA: 2s - loss: 8.4207 -
accuracy: 0.52 - ETA: 2s - loss: 8.9917 - accuracy: 0.51 - ETA: 2s - loss:
9.4588 - accuracy: 0.49 - ETA: 2s - loss: 9.9089 - accuracy: 0.48 - ETA: 2s -
loss: 10.3099 - accuracy: 0.468 - ETA: 1s - loss: 10.1493 - accuracy: 0.478 -
ETA: 1s - loss: 9.8978 - accuracy: 0.491 - ETA: 1s - loss: 9.6573 - accuracy:
0.50 - ETA: 1s - loss: 9.4289 - accuracy: 0.51 - ETA: 1s - loss: 9.2119 -
accuracy: 0.52 - ETA: 1s - loss: 9.0042 - accuracy: 0.53 - ETA: 1s - loss:
8.8052 - accuracy: 0.54 - ETA: 1s - loss: 8.9685 - accuracy: 0.53 - ETA: 1s -
loss: 9.1685 - accuracy: 0.52 - ETA: 1s - loss: 9.4219 - accuracy: 0.51 - ETA:
1s - loss: 9.3910 - accuracy: 0.51 - ETA: 1s - loss: 9.2041 - accuracy: 0.52 -
ETA: 1s - loss: 9.0049 - accuracy: 0.53 - ETA: 1s - loss: 8.8446 - accuracy:
0.53 - ETA: 1s - loss: 8.7600 - accuracy: 0.52 - ETA: 1s - loss: 8.6537 -
accuracy: 0.51 - ETA: 1s - loss: 8.5026 - accuracy: 0.52 - ETA: 1s - loss:
8.3355 - accuracy: 0.53 - ETA: 0s - loss: 8.1907 - accuracy: 0.54 - ETA: 0s -
loss: 8.0509 - accuracy: 0.54 - ETA: 0s - loss: 7.9009 - accuracy: 0.55 - ETA:
0s - loss: 7.7565 - accuracy: 0.56 - ETA: 0s - loss: 7.6310 - accuracy: 0.57 -
ETA: 0s - loss: 7.5101 - accuracy: 0.57 - ETA: 0s - loss: 7.3928 - accuracy:
0.58 - ETA: 0s - loss: 7.2786 - accuracy: 0.59 - ETA: 0s - loss: 7.1559 -
accuracy: 0.59 - ETA: 0s - loss: 7.0393 - accuracy: 0.60 - ETA: 0s - loss:
6.9384 - accuracy: 0.61 - ETA: 0s - loss: 6.8280 - accuracy: 0.61 - ETA: 0s -
loss: 6.7312 - accuracy: 0.62 - ETA: 0s - loss: 6.6365 - accuracy: 0.62 - ETA:
0s - loss: 6.5344 - accuracy: 0.63 - ETA: 0s - loss: 6.4473 - accuracy: 0.63 -
ETA: 0s - loss: 6.3525 - accuracy: 0.64 - ETA: 0s - loss: 6.2683 - accuracy:
0.64 - ETA: 0s - loss: 6.1863 - accuracy: 0.65 - 4s 178us/sample - loss: 6.1818
- accuracy: 0.6545
```

[101]: [6.181768329076739, 0.6544545]

[102]:
```
model_best.evaluate(X_test, y_test, verbose=False)
y_pred = model_best.predict(X_test)
cm = confusion_matrix(y_true=np.argmax(y_test, axis=1), y_pred=np.
 ↪argmax(y_pred, axis=1))
plot_confusion_matrix(cm, liquidos)
for i in range(4):
    print(f'{liquidos[i]} \t {cm[i,i]/cm[i,:].sum()} %')
```

```
Confusion matrix, without normalization
[[3498  617  711  174]
 [  12 1631  493 2864]
 [   0    4 3269 2727]
 [   0    0    0 6000]]
agua             0.6996 %
acucar           0.3262 %
aspartame        0.5448333333333333 %
sucralose        1.0 %
```



[ ]: