openmic

December 17, 2019

[1]:
```
import librosa as lb
import librosa.display
import pandas as pd
import scipy
import json
import numpy as np
import sklearn
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import os
from pylab import plot, show, figure, imshow, xlim, ylim, title
import matplotlib.pyplot as plt
import keras
from keras.utils import np_utils
from keras import layers
from keras import models
```

```
Using TensorFlow backend.
```

[2]:
```
#CONSTANTS

DATA_DIR = "openmic-2018/"
CATEGORY_COUNT = 8
```

[3]:
```
df = pd.read_csv('openmic-2018/openmic-2018-aggregated-labels.csv')
del df['relevance']
del df['num_responses']
```

[4]:
```
labels = df.values
labels
```

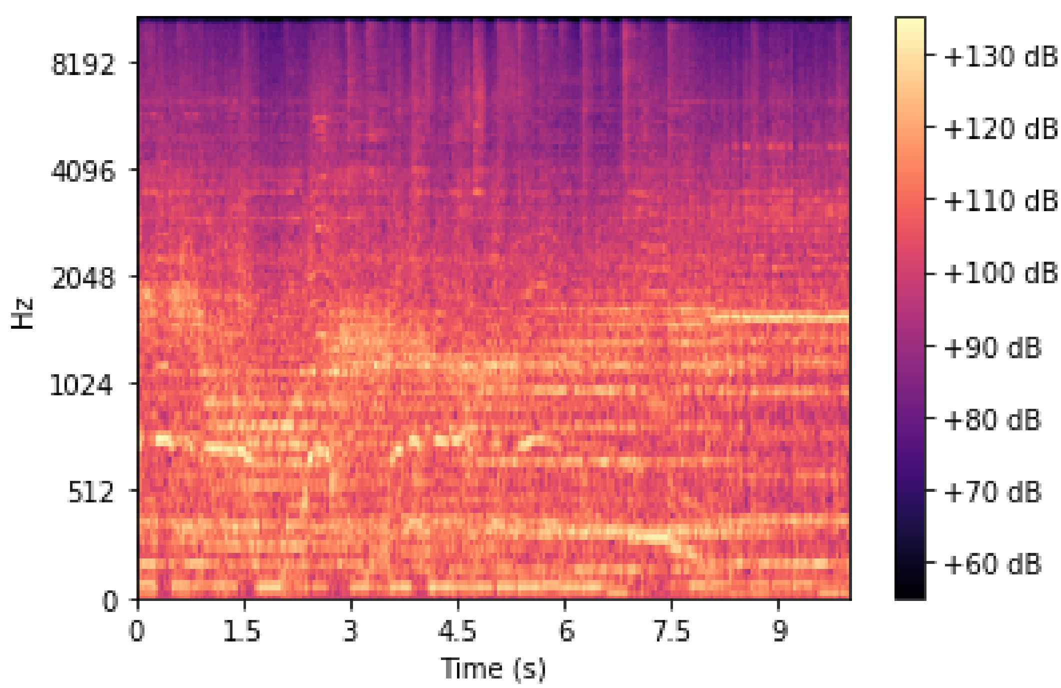
[4]:
```
array([['000046_3840', 'clarinet'],
       ['000046_3840', 'flute'],
       ['000046_3840', 'trumpet'],
       ...,
       ['155311_453120', 'saxophone'],
       ['155311_453120', 'trumpet'],
       ['155311_453120', 'trombone']], dtype=object)
```

[5]:
```
y, sr = lb.load(DATA_DIR + 'audio/000/000135_483840.ogg')
S = lb.feature.melspectrogram(y=y, sr=sr)

S_dB = lb.power_to_db(S, ref=0) # 10 * log10(S / ref)


print(y.shape)
print(sr)
print(S.shape)
print(S_dB.shape)
```

```
(220544,)
22050
(128, 431)
(128, 431)
```

[6]:
```
librosa.display.specshow(S_dB, x_axis='s', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
```

[6]: <matplotlib.colorbar.Colorbar at 0x234e4671278>



[7]:
```
OPENMIC = np.load(os.path.join(DATA_DIR, 'openmic-2018.npz'), allow_pickle=True)
print(list(OPENMIC.keys()))
```

```
['X', 'Y_true', 'Y_mask', 'sample_key']
```

[8]:
```
X, Y_true, Y_mask, sample_key = OPENMIC['X'], OPENMIC['Y_true'],␣
↪OPENMIC['Y_mask'], OPENMIC['sample_key']
#print(X.shape)
#X = []
#print(len(sample_key))
#for key in sample_key:
#    key_dir = key[:3]
#    y, sr = lb.load(DATA_DIR + 'audio/'+ key_dir + '/' + key + '.ogg')
#    X.append(lb.feature.melspectrogram(y=y, sr=sr))
#    print(len(X))
```

[9]:
```
with open(os.path.join(DATA_DIR, 'class-map.json'), 'r') as f:
    class_map = json.load(f)
```

[21]:
```
split_train, split_test, X_train, X_test, Y_true_train, Y_true_test,␣
↪Y_mask_train, Y_mask_test = train_test_split(sample_key, X, Y_true, Y_mask)
split_val, split_test, X_val, X_test, Y_true_val, Y_true_test, Y_mask_val,␣
↪Y_mask_test = train_test_split(split_test, X_test, Y_true_test, Y_mask_test,␣
↪test_size=0.5)

train_set = np.asarray(set(split_train))
test_set = np.asarray(set(split_test))
print('# Train: {},  # Val: {},  # Test: {}'.format(len(split_train),␣
↪len(split_test), len(split_val)))
```

```
# Train: 15000,  # Val: 2500,  # Test: 2500
```

[23]:
```
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
(15000, 10, 128)
(2500, 10, 128)
(2500, 10, 128)
```

[27]:
```
THRESHOLD = 0.3

# This dictionary will include the classifiers for each model
mymodels = dict()

# We'll iterate over all istrument classes, and fit a model for each one
# After training, we'll print a classification report for each instrument
for instrument in class_map:

    # Map the instrument name to its column number
    inst_num = class_map[instrument]
```

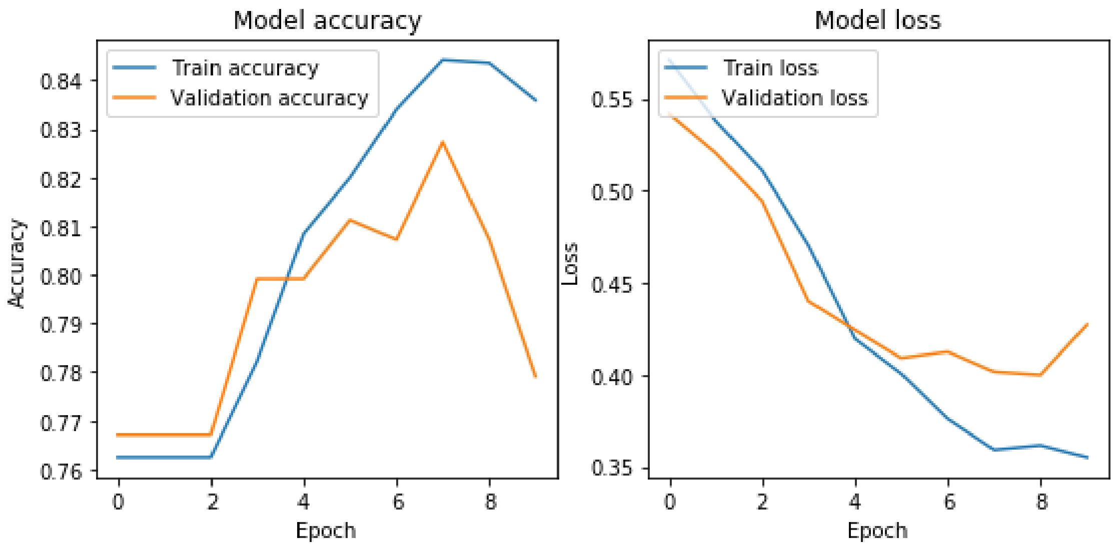
```
# Step 1: sub-sample the data

# First, we need to select down to the data for which we have annotations
# This is what the mask arrays are for
train_inst = Y_mask_train[:, inst_num]
val_inst = Y_mask_val[:, inst_num]
test_inst = Y_mask_test[:, inst_num]

# Here, we're using the Y_mask_train array to slice out only the training␣
↪examples
# for which we have annotations for the given class
X_train_inst = X_train[train_inst]
X_val_inst = X_val[val_inst]

# Step 3: simplify the data by averaging over time

# Let's arrange the data for a sklearn Random Forest model
# Instead of having time-varying features, we'll summarize each track by␣
↪its mean feature vector over time
X_train_inst_sklearn = np.mean(X_train_inst, axis=1)

# Again, we slice the labels to the annotated examples
# We thresold the label likelihoods at 0.5 to get binary labels
Y_true_train_inst = Y_true_train[train_inst, inst_num] >= THRESHOLD
Y_true_val_inst = Y_true_val[val_inst, inst_num] >= THRESHOLD


# Repeat the above slicing and dicing but for the test set
X_test_inst = X_test[test_inst]
X_test_inst_sklearn = np.mean(X_test_inst, axis=1)
Y_true_test_inst = Y_true_test[test_inst, inst_num] >= THRESHOLD


X_train_inst = X_train_inst.astype('float32')
X_val_inst = X_val_inst.astype('float32')
X_train_inst_sklearn = X_train_inst_sklearn.astype('float32')
X_train_inst_sklearn = lb.util.normalize(X_train_inst_sklearn)
# X_train_inst = S_dB
print(X_train_inst.shape)
shape = X_train_inst.shape
X_train_inst = X_train_inst.reshape(shape[0],1, shape[1], shape[2])
shape = X_val_inst.shape
X_val_inst = X_val_inst.reshape(shape[0],1, shape[1], shape[2])
shape = X_test_inst.shape
X_test_inst = X_test_inst.reshape(shape[0],1, shape[1], shape[2])
#X_train_inst = X_train_inst.reshape(1,1,431,128)
print(X_train_inst.shape)
```

```
print(Y_true_train_inst[0])
# Step 3.
# Initialize a new classifier
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
model = models.Sequential()

# model.add(layers.Conv2D(filters=8,kernel_size=(3,3),activation='relu',␣
↪input_shape=(10,128,1,)))
model.
↪add(Conv2D(input_shape=(1,10,128),data_format="channels_first",filters=64,kernel_size=(3,3)
↪activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",␣
↪activation="relu"))
model.add(MaxPool2D(pool_size=(3,3),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same",␣
↪activation="relu"))
model.add(layers.Flatten())
model.add(layers.Dense(units=4096, activation='relu'))
model.add(layers.Dense(units=4096, activation='relu'))
model.add(layers.Dense(units=1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=keras.optimizers.Adam(lr=0.00001),
               metrics = ['accuracy'])

# model.summary()
```

```
    # Step 4.
    history = model.fit(X_train_inst,Y_true_train_inst , epochs=10,␣
↪batch_size=64, validation_data=(X_val_inst,Y_true_val_inst))

    plt.figure(figsize=(9,4))

    plt.subplot(1,2,1)
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train accuracy', 'Validation accuracy'], loc='upper left')

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train loss', 'Validation loss'], loc='upper left')
    plt.show()

    loss, acc = model.evaluate(X_test_inst, Y_true_test_inst)
    print('Test loss: {}'.format(loss))
    print('Test accuracy: {:.2%}'.format(acc))

    # Step 5.
    # Finally, we'll evaluate the model on both train and test
    Y_pred_train = model.predict(X_train_inst)
    Y_pred_test = model.predict(X_test_inst)
    Y_pred_train_bool = Y_pred_train > THRESHOLD #THRESHOLD (should be lower␣
↪than 0.5)
    Y_pred_test_bool = Y_pred_test > THRESHOLD #THRESHOLD (should be lower than␣
↪0.5)
    print(Y_pred_train[0])
    print('-' * 52)
    print(instrument)
    print('\tTRAIN')
    print(classification_report(Y_true_train_inst, Y_pred_train_bool))

    print(Y_true_train_inst[3])
    print(Y_pred_train[3])
    print('\tTEST')
    print(classification_report(Y_true_test_inst, Y_pred_test_bool))
    sum = 0
#   for i, prob in enumerate(Y_pred_train):
```

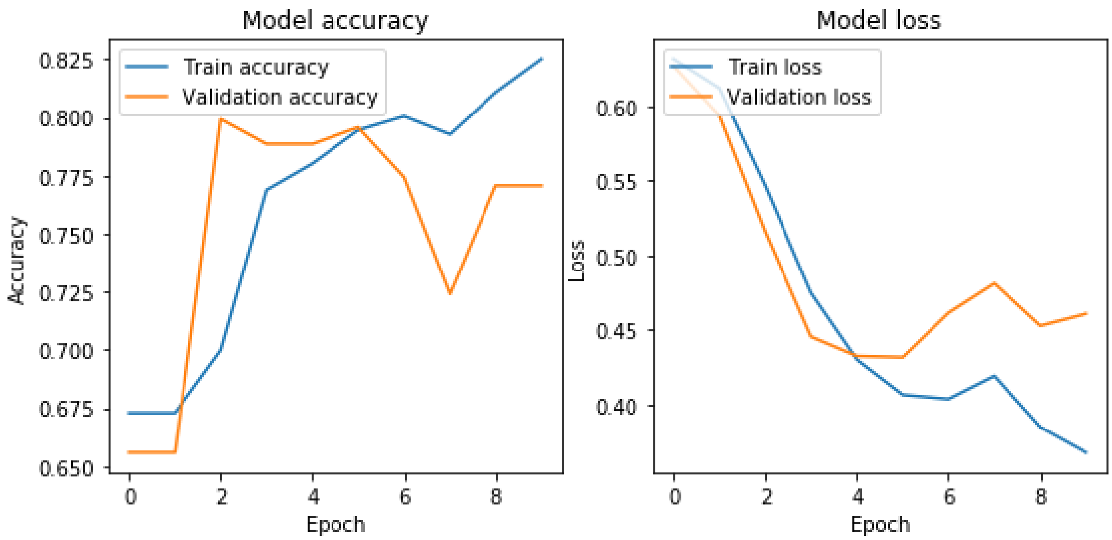
```
#       print (i)
#       print (prob)
#       sum += prob
#   print(sum)
    # Store the classifier in our dictionary
    mymodels[instrument] = model
```

```
(1566, 10, 128)
(1566, 1, 10, 128)
False
Train on 1566 samples, validate on 249 samples
Epoch 1/10
1566/1566 [==============================] - 90s 57ms/step - loss: 0.5710 - acc:
0.7625 - val_loss: 0.5417 - val_acc: 0.7671
Epoch 2/10
1566/1566 [==============================] - 87s 56ms/step - loss: 0.5380 - acc:
0.7625 - val_loss: 0.5206 - val_acc: 0.7671
Epoch 3/10
1566/1566 [==============================] - 88s 56ms/step - loss: 0.5110 - acc:
0.7625 - val_loss: 0.4943 - val_acc: 0.7671
Epoch 4/10
1566/1566 [==============================] - 88s 56ms/step - loss: 0.4703 - acc:
0.7822 - val_loss: 0.4399 - val_acc: 0.7992
Epoch 5/10
1566/1566 [==============================] - 87s 55ms/step - loss: 0.4196 - acc:
0.8084 - val_loss: 0.4243 - val_acc: 0.7992
Epoch 6/10
1566/1566 [==============================] - 86s 55ms/step - loss: 0.4006 - acc:
0.8199 - val_loss: 0.4090 - val_acc: 0.8112
Epoch 7/10
1566/1566 [==============================] - 86s 55ms/step - loss: 0.3760 - acc:
0.8340 - val_loss: 0.4124 - val_acc: 0.8072
Epoch 8/10
1566/1566 [==============================] - 86s 55ms/step - loss: 0.3591 - acc:
0.8442 - val_loss: 0.4015 - val_acc: 0.8273
Epoch 9/10
1566/1566 [==============================] - 86s 55ms/step - loss: 0.3615 - acc:
0.8436 - val_loss: 0.3998 - val_acc: 0.8072
Epoch 10/10
1566/1566 [==============================] - 85s 55ms/step - loss: 0.3551 - acc:
0.8359 - val_loss: 0.4271 - val_acc: 0.7791
```




```
256/256 [==============================] - 2s 10ms/step
Test loss: 0.4729207083582878
Test accuracy: 75.00%
[0.7572417]
----------------------------------------------------
accordion
        TRAIN
              precision    recall  f1-score   support

       False       0.96      0.71      0.82      1194
        True       0.49      0.92      0.64       372

    accuracy                           0.76      1566
   macro avg       0.73      0.81      0.73      1566
weighted avg       0.85      0.76      0.77      1566

False
[0.6689949]
        TEST
              precision    recall  f1-score   support

       False       0.90      0.66      0.76       197
        True       0.40      0.75      0.52        59

    accuracy                           0.68       256
   macro avg       0.65      0.70      0.64       256
weighted avg       0.78      0.68      0.70       256

(1669, 10, 128)
```

```
(1669, 1, 10, 128)
True
Train on 1669 samples, validate on 279 samples
Epoch 1/10
1669/1669 [==============================] - 102s 61ms/step - loss: 0.6316 -
acc: 0.6729 - val_loss: 0.6269 - val_acc: 0.6559
Epoch 2/10
1669/1669 [==============================] - 93s 56ms/step - loss: 0.6120 - acc:
0.6729 - val_loss: 0.5936 - val_acc: 0.6559
Epoch 3/10
1669/1669 [==============================] - 93s 56ms/step - loss: 0.5465 - acc:
0.6998 - val_loss: 0.5156 - val_acc: 0.7993
Epoch 4/10
1669/1669 [==============================] - 93s 56ms/step - loss: 0.4749 - acc:
0.7687 - val_loss: 0.4455 - val_acc: 0.7885
Epoch 5/10
1669/1669 [==============================] - 93s 55ms/step - loss: 0.4302 - acc:
0.7801 - val_loss: 0.4325 - val_acc: 0.7885
Epoch 6/10
1669/1669 [==============================] - 101s 60ms/step - loss: 0.4065 -
acc: 0.7945 - val_loss: 0.4316 - val_acc: 0.7957
Epoch 7/10
1669/1669 [==============================] - 101s 61ms/step - loss: 0.4036 -
acc: 0.8005 - val_loss: 0.4614 - val_acc: 0.7742
Epoch 8/10
1669/1669 [==============================] - 107s 64ms/step - loss: 0.4192 -
acc: 0.7927 - val_loss: 0.4812 - val_acc: 0.7240
Epoch 9/10
1669/1669 [==============================] - 108s 65ms/step - loss: 0.3849 -
acc: 0.8107 - val_loss: 0.4528 - val_acc: 0.7706
Epoch 10/10
1669/1669 [==============================] - 102s 61ms/step - loss: 0.3679 -
acc: 0.8250 - val_loss: 0.4608 - val_acc: 0.7706
```




```
270/270 [==============================] - 3s 10ms/step
Test loss: 0.4562561829884847
Test accuracy: 77.78%
[0.9145529]
----------------------------------------------------
banjo
        TRAIN
              precision    recall  f1-score   support

       False       0.90      0.85      0.87      1123
        True       0.72      0.80      0.76       546

    accuracy                           0.83      1669
   macro avg       0.81      0.83      0.82      1669
weighted avg       0.84      0.83      0.84      1669

False
[0.00839368]
        TEST
              precision    recall  f1-score   support

       False       0.86      0.79      0.82       180
        True       0.64      0.73      0.68        90

    accuracy                           0.77       270
   macro avg       0.75      0.76      0.75       270
weighted avg       0.78      0.77      0.78       270

(1401, 10, 128)
```

```
(1401, 1, 10, 128)
False
Train on 1401 samples, validate on 258 samples
Epoch 1/10
```

```
        ␣
↪---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call␣
↪last)

        <ipython-input-27-0129277914b6> in <module>
         91     # model.summary()
         92      # Step 4.
    ---> 93      history = model.fit(X_train_inst,Y_true_train_inst , epochs=10,␣
↪batch_size=64, validation_data=(X_val_inst,Y_true_val_inst))
         94
         95      plt.figure(figsize=(9,4))


        ~\Documents\Conda\lib\site-packages\keras\engine\training.py in␣
↪fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split,␣
↪validation_data, shuffle, class_weight, sample_weight, initial_epoch,␣
↪steps_per_epoch, validation_steps, **kwargs)
       1037                                          initial_epoch=initial_epoch,
       1038                                         ␣
↪steps_per_epoch=steps_per_epoch,
    -> 1039                                         ␣
↪validation_steps=validation_steps)
       1040
       1041     def evaluate(self, x=None, y=None,


        ~\Documents\Conda\lib\site-packages\keras\engine\training_arrays.py in␣
↪fit_loop(model, f, ins, out_labels, batch_size, epochs, verbose, callbacks,␣
↪val_f, val_ins, shuffle, callback_metrics, initial_epoch, steps_per_epoch,␣
↪validation_steps)
        197                     ins_batch[i] = ins_batch[i].toarray()
        198
    --> 199                 outs = f(ins_batch)
        200                 outs = to_list(outs)
        201                 for l, o in zip(out_labels, outs):


        ~\Documents\Conda\lib\site-packages\keras\backend\tensorflow_backend.py␣
↪in __call__(self, inputs)
```

```
   2713                     return self._legacy_call(inputs)
   2714
-> 2715                 return self._call(inputs)
   2716             else:
   2717                 if py_any(is_tensor(x) for x in inputs):


    ~\Documents\Conda\lib\site-packages\keras\backend\tensorflow_backend.py␣
↪in _call(self, inputs)
   2673                 fetched = self._callable_fn(*array_vals,␣
↪run_metadata=self.run_metadata)
   2674             else:
-> 2675                 fetched = self._callable_fn(*array_vals)
   2676             return fetched[:len(self.outputs)]
   2677


    ~\Documents\Conda\lib\site-packages\tensorflow\python\client\session.py␣
↪in __call__(self, *args, **kwargs)
   1437               ret = tf_session.TF_SessionRunCallable(
   1438                   self._session._session, self._handle, args, status,
-> 1439                   run_metadata_ptr)
   1440             if run_metadata:
   1441               proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


    KeyboardInterrupt:
```

[ ]:
```
print(X_train_inst_sklearn)
print(Y_pred_train)
```

[ ]: