ДОДАТОК А.
ПРОГРАМНА РЕАЛІЗАЦІЯ

```
import librosa
import librosa.feature
import librosa.display
import glob
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils.np_utils import to_categorical

def display_mfcc(song):
    y, _ = librosa.load(song)
    mfcc = librosa.feature.mfcc(y)
    plt.figure(figsize=(10, 4))
    librosa.display.specshow(mfcc,          x_axis='time',
y_axis='mel')    plt.colorbar()
    plt.title(song)
    plt.tight_layout()
    plt.show()

display_mfcc('genres/disco/disco.00035.au')

def extract_features_song(f):
    y, _ = librosa.load(f
    # get Mel-frequency cepstral coefficients
    mfcc = librosa.feature.mfcc(y)
```

```
    # normalize values between -1,1 (divide by max)
    mfcc /= np.amax(np.absolute(mfcc)
    return np.ndarray.flatten(mfcc)[:25000]

def generate_features_and_labels():
    all_features = []
    all_labels = []


    genres = ['blues', 'classical', 'country', 'disco',
'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
    for genre in genres:
        sound_files = glob.glob('genres/'+genre+'/*.au')
        print('Processing %d songs in %s genre...' %
(len(sound_files), genre))
        for f in sound_files:
            features = extract_features_song(f)
            all_features.append(features)
            all_labels.append(genre)

    # convert labels to one-hot encoding
    label_uniq_ids, label_row_ids = np.unique(all_labels,
return_inverse=True)
    label_row_ids = label_row_ids.astype(np.int32,
copy=False)
    onehot_labels = to_categorical(label_row_ids,
len(label_uniq_ids))
    return np.stack(all_features), onehot_labels

features, labels = generate_features_and_labels()
```

```
print(np.shape(features))
print(np.shape(labels))

training_split = 0.8

# last column has genre, turn it into unique ids
alldata = np.column_stack((features, labels))

np.random.shuffle(alldata)
splitidx = int(len(alldata) * training_split)
train, test = alldata[:splitidx,:], alldata[splitidx:,:]

print(np.shape(train))
print(np.shape(test))

train_input = train[:,:-10]
train_labels = train[:,-10:]

test_input = test[:,:-10]
test_labels = test[:,-10:]

print(np.shape(train_input))
print(np.shape(train_labels))

model = Sequential([
    Dense(100, input_dim=np.shape(train_input)[1]),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
    ])
```

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

model.fit(train_input,      train_labels,      epochs=10,
batch_size=32,
          validation_split=0.2)

loss,   acc   =   model.evaluate(test_input,   test_labels,
batch_size=32)

print("Done!")
print("Loss: %.4f, accuracy: %.4f" % (loss, acc))
```