Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«Южно-Уральский государственный университет»

(Национальный исследовательский университет)

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

ОТЧЕТ

по практической работе №4

|  |  |  |  |
| --- | --- | --- | --- |
| Работа со звуком | | | |
|  | | | |
|  |  | Руководитель: | |
|  |  | | \_\_\_\_\_\_\_\_\_\_\_К.Ю. Никольская  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.  Автор работы  студент группы КЭ-220  \_\_\_\_\_\_\_\_\_\_\_\_А.В.Витомсков  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

Челябинск 2020

1Развернуть у себя и показать преподавателю работоспособность, ответив на вопросы понимания кода:  
<https://towardsdatascience.com/urban-sound-classification-using-neural-networks-9b6fcd8a9150>

Подключаем библиотеки.

import  IPython.display  as  ipd

import  os

import  numpy  as  np

import  pandas  as  pd

import  matplotlib.pyplot  as  plt

import  librosa

from  tqdm  import tqdm

from  sklearn.preprocessing  import StandardScaler

from  keras.models  import Sequential

from  keras.layers  import Dense, Dropout, Activation

from  keras.optimizers  import Adam

Монтируем гугл-диск

from google.colab import drive

drive.mount('/gdrive')

Читаем метки классов

data = pd.read\_csv ('/gdrive/My Drive/urban/train/train.csv')

data.head () # Чтобы увидеть набор данных

Используя библиотеку librosa, извлекаем четыре функции из аудиофайлов. Этими характеристиками являются мел-кепстральные коэффициенты (MFCC), тоннец, спектрограмма в масштабе Mel и хроматограмма из формы волны.

Проходим по всем файлам.

Извлекаем из каждого мел-кепстральные коэффициенты, находим среднее значение.

Считаем тональные центроиды (тоннец), находим средние значения для файла.

Считаем мел-спектрограмму, вычисляем среднее

Считаем оконное преобразование Фурье, на его основе вычисляется хроматограмма. Находим среднее.

mfc=[]

chr=[]

me=[]

ton=[]

lab=[]

for i in tqdm(range(len(data))):

    f\_name='/gdrive/My Drive/urban/train/Train/'+str(data.ID[i])+'.wav'

    X, s\_rate = librosa.load(f\_name, res\_type='kaiser\_fast')

    mf = np.mean(librosa.feature.mfcc(y=X, sr=s\_rate).T,axis=0)

    mfc.append(mf)

    l=data.Class[i]

    lab.append(l)

    try:

        t =    np.mean(librosa.feature.tonnetz(

                       y=librosa.effects.harmonic(X),

                       sr=s\_rate).T,axis=0)

        ton.append(t)

    except:

        print(f\_name)

    m = np.mean(librosa.feature.melspectrogram(X, sr=s\_rate).T,axis=0)

    me.append(m)

    s = np.abs(librosa.stft(X))

    c = np.mean(librosa.feature.chroma\_stft(S=s, sr=s\_rate).T,axis=0)

    chr.append(c)

Сохраняем результат

mfcc = pd.DataFrame(mfc)

mfcc.to\_csv('/gdrive/My Drive/urban/mfc.csv', index=False)

chrr = pd.DataFrame(chr)

chrr.to\_csv('/gdrive/My Drive/urban/chr.csv', index=False)

mee = pd.DataFrame(me)

mee.to\_csv('/gdrive/My Drive/urban/me.csv', index=False)

tonn = pd.DataFrame(ton)

tonn.to\_csv('/gdrive/My Drive/urban/ton.csv', index=False)

la = pd.DataFrame(lab)

la.to\_csv('/gdrive/My Drive/urban/labels.csv', index=False)

Загружаем

mfcc = pd.read\_csv('/gdrive/My Drive/urban/mfc.csv', index\_col=None)

chrr = pd.read\_csv('/gdrive/My Drive/urban/chr.csv', index\_col=None)

mee = pd.read\_csv('/gdrive/My Drive/urban/me.csv', index\_col=None)

tonn = pd.read\_csv('/gdrive/My Drive/urban/ton.csv', index\_col=None)

la = pd.read\_csv('/gdrive/My Drive/urban/labels.csv', index\_col=None)

mfc = mfcc.values.tolist()

chr = chrr.values.tolist()

me = mee.values.tolist()

lab = la.values.tolist()

ton = tonn.values.tolist()

Все фичи объединяются в один массив.

features = []

for i in range(len(ton)):

    features.append(np.concatenate((me[i], mfc[i],

                ton[i], chr[i]), axis=0))

Метки перекодируются из категориальной переменной в непрерывную.

Кодирование One-Hot - это широко используемый метод преобразования категориальной входной переменной в непрерывную переменную. Для каждого присутствующего уровня будет создана одна новая переменная. Наличие уровня представлено 1, а отсутствие - 0.

la = pd.get\_dummies(la)

label\_columns=la.columns #To get the classes

target = la.to\_numpy() #Convert labels to numpy array

Нормализуем входные данные.

tran = StandardScaler()

features\_train = tran.fit\_transform(features)

Разделяем на 3 набора: обучающий, валидационный, тестовый.

feat\_train=features\_train[:4434]

target\_train=target[:4434]

y\_train=features\_train[4434:5330]

y\_val=target[4434:5330]

test\_data=features\_train[5330:]

test\_label=lab[5330:]

Определение метрик для валидационного набора

from keras import backend as K

# Метрики

def precision(y\_true, y\_pred):

    """Precision metric.

    Only computes a batch-wise average of precision.

    Computes the precision, a metric for multi-label classification of

    how many selected items are relevant.

    """

    true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

    predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

    precision = true\_positives / (predicted\_positives + K.epsilon())

    return precision

def recall(y\_true, y\_pred):

    """Recall metric.

    Only computes a batch-wise average of recall.

    Computes the recall, a metric for multi-label classification of

    how many relevant items are selected.

    """

    true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

    possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

    recall = true\_positives / (possible\_positives + K.epsilon())

    return recall

def f1(y\_true, y\_pred):

    def recall(y\_true, y\_pred):

        """Recall metric.

        Only computes a batch-wise average of recall.

        Computes the recall, a metric for multi-label classification of

        how many relevant items are selected.

        """

        true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

        possible\_positives = K.sum(K.round(K.clip(y\_true, 0, 1)))

        recall = true\_positives / (possible\_positives + K.epsilon())

        return recall

    def precision(y\_true, y\_pred):

        """Precision metric.

        Only computes a batch-wise average of precision.

        Computes the precision, a metric for multi-label classification of

        how many selected items are relevant.

        """

        true\_positives = K.sum(K.round(K.clip(y\_true \* y\_pred, 0, 1)))

        predicted\_positives = K.sum(K.round(K.clip(y\_pred, 0, 1)))

        precision = true\_positives / (predicted\_positives + K.epsilon())

        return precision

    precision = precision(y\_true, y\_pred)

    recall = recall(y\_true, y\_pred)

    return 2 \* ((precision \* recall) / (precision + recall + K.epsilon()))

Определяем модель.

Модель нейросети состоит из 4-х слоёв. Внутренние слои «прорежены».

Минимизируем функцию потерь – кроссэнтропию.

model = Sequential()

model.add(Dense(166, input\_shape=(166,), activation = 'relu'))

model.add(Dense(256, activation = 'relu'))

model.add(Dropout(0.6))

model.add(Dense(128, activation = 'relu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation = 'softmax'))

model.compile(loss='categorical\_crossentropy', metrics=['accuracy', precision, recall, f1], optimizer='adam')

print(type(feat\_train))

print(type(target\_train))

Обучение

history = model.fit(feat\_train, target\_train, batch\_size=64, epochs=100, validation\_data=(y\_train, y\_val))

Результат тестирования

prediction=[]

for i in predict:

    j=label\_columns[i]

    prediction.append(j)

print(prediction)

print(lab)

k=0

for i, j in zip(test\_label,prediction):

    #print (i[0])

    #print (j[2:])

    if i[0]==j[2:]:

        k=k+1

print(len(prediction))

print(k)

88/103 файлов классифицированы верно.