МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Анализ звука и голоса»

Тема: «Классификация произвольных звуков»

Студент гр. 6304 _____ Белоусов Е. О.

Преподаватель _____ Рыбин С. В.

Санкт-Петербург

2021

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ И КРИТЕРИИ ОЦЕНКИ

Цель работы: классифицировать произвольные звуки из набора данных Freesound (ссылка на соревнование на платформе Kaggle: https://www.kaggle.com/c/freesound-audio-tagging/overview).

Критерии оценки:

1. Если результат больше 0.76 (точность), то оценка 10 баллов.
2. Если результат не меньше 0.7 (точность), то оценка 8 баллов.
3. Если результат не меньше 0.33 (точность), то оценка 6 баллов.

ХОД РАБОТЫ

Задача классификации

Задача многоклассовой классификации — это задача классификации данных на более чем два класса. Для решения задачи классификации с многими метками аналогично можно применять алгоритмы, которые естественным образом преобразуются в многоклассовые (деревья решений, логистическая регрессия, нейронные сети и другие).

Набор данных «Freesound»

Некоторые звуки отчетливы и мгновенно узнаваемы, например детский смех или бренчание гитары. Другие звуки нечеткие, и их трудно определить.

Более того, мы часто слышим смесь звуков, создающих атмосферу – например, шум строителей, шум транспорта за дверью, смешанный с громким смехом из комнаты, тиканье часов на стене.

Частично из-за огромного количества звуков, которые мы слышим, не существует надежных автоматических универсальных систем разметок аудио. В настоящее время требуется много ручных усилий для таких задач, как аннотирование коллекций звуков и предоставление титров для неречевых событий в аудиовизуальном контенте.

Чтобы решить эту проблему, Freesound (инициатива MTG-UPF, которая поддерживает совместную базу данных с более чем 370000 звуков, лицензированных Creative Commons) и команда Google Research Machine Perception (создатели AudioSet, крупномасштабного набора данных вручную аннотированных звуковых событий с более чем 500 классы) объединились для разработки набора данных.

Создание и настройка модели нейронной сети

Для работы с моделью искусственной нейронной сети (ИНС) используется язык программирования Python 3.7.6. Используются следующие библиотеки:

- Pandas – для загрузки набора данных;
- Scikit-learn – для предварительной обработки данных и оценки работы модели ИНС;
- Librosa – для работы со звуковыми файлами;

- Keras (в составе библиотеки TensorFlow) – для создания и настройки модели нейронной сети.

Свёрточная нейронная сеть

Оператор свертки составляет основу свёрточного слоя (convolutional layer) в CNN. Слой состоит из определенного количества ядер (с аддитивными составляющими смещения для каждого ядра) и вычисляет свертку выходного изображения предыдущего слоя с помощью каждого из ядер, каждый раз прибавляя составляющую смещения. В конце концов ко всему выходному изображению может быть применена функция активации. Обычно входной поток для свёрточного слоя состоит из d каналов, например, RGB для входного слоя, и в этом случае ядра тоже расширяют таким образом, чтобы они также состояли из d каналов; получается следующая формула для одного канала выходного изображения свёрточного слоя, где K — ядро, а b — составляющая смещения:

$$conv(I\ ,K)_{xy} = \sigma\left(b + \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k=1}^{d} K_{ijk} \times I_{x+i-1,y+i-1,k}\right)$$

Стоит также отметить, что хотя сверточный слой сокращает количество параметров по сравнению с полносвязным слоем, он использует больше гиперпараметров — параметров, выбираемых до начала обучения.

В частности, выбираются следующие гиперпараметры:

- Глубина (depth) — сколько ядер и коэффициентов смещения будет задействовано в одном слое (в данной работе, 32 ядра на первом свёрточном слое и 64 на втором);
- Высота (height) и ширина (width) каждого ядра (в данной работе исследовались результаты при размере ядра 3x3 и 5x5);
- Шаг (stride) — на сколько смещается ядро на каждом шаге при вычислении следующего пикселя результирующего изображения. Обычно его принимают равным 1, и чем больше его значение, тем меньше размер выходного изображения.

Приступим к созданию и настройке модели ИНС. Основным строительным блоком нейронных сетей является слой (или уровень), модуль обработки данных,

который можно рассматривать как фильтр для данных. Он принимает некоторые данные и выводит их в более полезной форме. В частности, слои извлекают представления из подаваемых в них данных, которые, как мы надеемся, будут иметь больше смысла для решаемой задачи. Фактически методика глубокого обучения заключается в объединении простых слоев, реализующих некоторую форму поэтапной очистки данных.

Чтобы подготовить сеть к обучению, нужно настроить еще три параметра для этапа компиляции:

- функцию потерь (loss), которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении;
- оптимизатор (optimizer) — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь (в данном случае, Adam – adaptive moment estimation);
- метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность классификации (accuracy).

Листинг 1 – Создание и настройка модели ИНС

```
def get_2d_conv_model(config):

    nclass = config.n_classes

    inp = Input(shape=(config.dim[0],config.dim[1],1))

    x = Convolution2D(32, (4,10), padding="same")(inp)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Convolution2D(32, (4,10), padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    x = MaxPool2D()(x)

    x = Flatten()(x)
    x = Dense(64)(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    out = Dense(nclass, activation=softmax)(x)
```

```
    model = models.Model(inputs=inp, outputs=out)
    opt = optimizers.Adam(config.learning_rate)

    model.compile(optimizer=opt, loss=losses.categorical_crossentropy,
metrics=['acc'])
    return model
```

Классификационные признаки и MFCC

Для решения задачи автоматической классификации объектов по их аудиоданным необходимо выбрать и сформировать набор классификационных признаков, на основании которых будет работать разрабатываемый классификатор. Поскольку в рассматриваемой задаче вся информация об объектах заключена только в их звуковых сигналах, КП необходимо выделять из сигналов в процессе проведения анализа акустических данных.

Мел-частотные кепстральные коэффициенты (Mel-frequency cepstral coefficients, MFCC) наиболее часто применяются в задачах, связанных с анализом человеческой речи.

Мел – некоторая психофизическая единица высоты звука, построенная на восприятии звука человеческими органами слуха. Высота звука, воспринимаемая человеком, описывается следующей нелинейной зависимостью от частоты:

$$m = 1125 \ln\left(1 + \frac{f}{700}\right), \quad (1)$$

где f – частота звука в Герцах.

Кепстром называется функция обратного преобразования Фурье от логарифма спектра мощности сигнала, описываемая следующим выражением:

$$C(q) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \ln|S(\omega)|^2 \, e^{i\omega q} d\omega,$$

где $S(\omega)$ – спектр входного сигнала; q – аргумент кепстрального времени (встречается наименование «quefrency»). Кепстр в любой момент q зависит от функции $s(t)$ (заданной при $t \in (-\infty; \infty)$) входного сигнала со спектром $S(\omega)$.

Акустические признаки MFCC строятся по следующему алгоритму:

1. Разбиение входного звукового сигнала на временные окна (фрэймы) малой длины (в среднем, 20 мс), с фиксированным перекрытием (в среднем, 10 мс).

2. Предварительная фильтрация при помощи фильтра с конечной импульсной характеристикой (КИХ-фильтр), описываемая следующим выражением:

$$y_t = x_t - b \cdot x_{t-1},$$

где y_t – сигнал после фильтрации; x_t – фильтруемый сигнал; b – коэффициент фильтрации, как правило, равный 0,95.

3. Применение дискретного преобразования Фурье (DFT):

$$F_k = \sum_{t=0}^{T-1} \omega_t \cdot y_t e^{-\frac{2\pi i}{T} kt} \text{ при } k = 0, \dots, \frac{T}{2},$$

где T – количество отсчётов в рассматриваемом фрейме; ω_t – весовая оконная функция, применяемая для уменьшения краевых эффектов, которые возникают в результате 1 этапа данного алгоритма. Наиболее популярные оконные функции: окно Хэмминга, окно Блэкмана, окно Хана.

4. Расчёт логарифма энергии спектра для набора треугольных Мел-частотных фильтров (рисунок 1):

$$E_s^{Mel} = \log\left(\sum_{k=0}^{\frac{T}{2}} |F_k| H_s^{Mel}(f_k)\right) \text{ при } s = 0, \dots, M-1,$$

где M – количество треугольных Мел-частотных фильтров; $H_s^{Mel}(f)$ – s-ый треугольный Мел-частотный фильтр, рассчитываемый следующим образом:

$$H_s^{Mel}(f) = \begin{cases} 0, & m(f) < m_{begin}^s \\ \dfrac{m(f) - m_{begin}^s}{m_{center}^s - m_{begin}^s}, & m_{begin}^s \le m(f) < m_{center}^s \\ \dfrac{m_{end}^s - m(f)}{m_{end}^s - m_{center}^s}, & m_{center}^s \le m(f) < m_{end}^s \\ 0, & m(f) \ge m_{end}^s \end{cases},$$

где $m(f)$ – частота в масштабе Мел-шкалы (1); m_{begin}^s, m_{center}^s, m_{end}^s – начало, середина, конец треугольного окна Мел-частотного фильтра, значения которых определяются следующими выражениями:

$$m_{begin}^s = m(f_{low}) + s \cdot \frac{m(f_{high}) - m(f_{low})}{M+1};$$

$$m_{end}^s = m(f_{low}) + (s+2) \cdot \frac{m(f_{high}) - m(f_{low})}{M+1};$$

$$m^s_{center} = \frac{1}{2}\left(m^s_{begin} + m^s_{end}\right),$$

где f_{low} и f_{high} – границы рассматриваемого частотного диапазона.

5. Вычисление дискретного косинусного преобразования для энергий, вычисленных на предыдущем этапе:

$$C_l = \sum_{s=0}^{M-1} E_s \cos\left(l\left(s + \frac{1}{2}\right)\frac{\pi}{M}\right) \text{ при } l = 0, \dots, M-1. \quad (2)$$

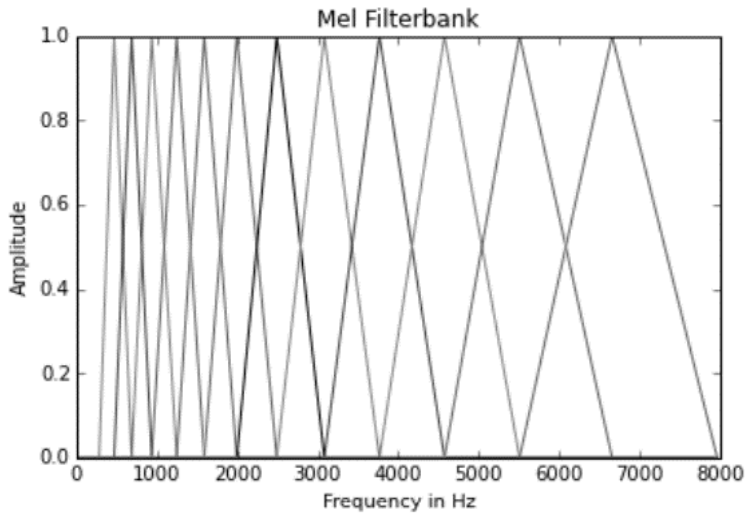

Рисунок 1 – Набор 12 треугольных Мел-частотных фильтров для звукового сигнала частотой от 300 до 8000 Гц

Использование MFCC позволяет сократить представление одного фрейма входного звукового сигнала до сравнительно небольшого количества коэффициентов (рисунок 2), каждый из которых вносит значительный вклад в конечный спектр. Как правило, для работы используются несколько (от 12 до 20) первых коэффициентов, полученных на 5 этапе алгоритма (2).

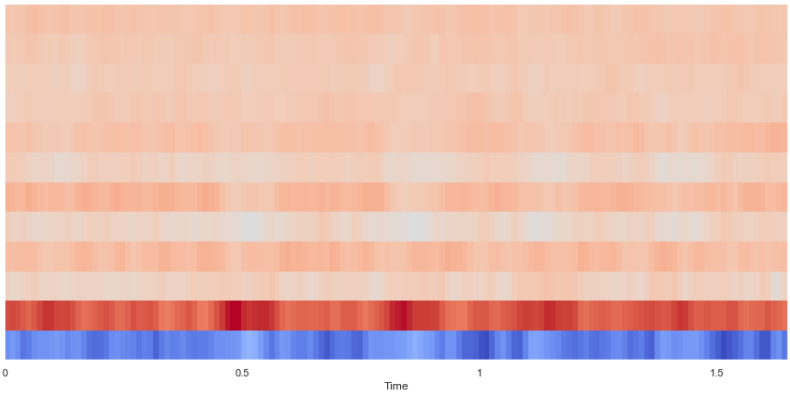

Рисунок 2 – Представление звукового сигнала в виде набора 12 Мел-частотных кепстральных коэффициентов

Стоит отметить, что есть возможность интерпретировать только первые два коэффициента:

1. C_0 – мощность всего частотного диапазона;

2. C_1 – баланс между низко- и высокочастотными компонентами в данном фрейме.

Остальные коэффициенты неинтерпретируемы, так как содержат мелкие детали спектра звукового сигнала, что также видно на рисунке 2 (полосы бледного цвета).

Обучение нейронной сети

Теперь можно начинать обучение сети, для чего, в случае использования библиотеки Keras, достаточно вызвать метод fit сети — он пытается адаптировать модель под обучающие данные. В процессе обучения отображаются четыре величины: потери сети на обучающих данных и точность сети на обучающих данных, а также потери и точность на данных, не участвовавших в обучении.

Количество эпох равно 50, папок в кросс-валидации – 10.

Листинг 2 – Создание и обучение модели ИНС

```
# Для кросс-валидации используется StratifiedKFold - разновдность KFold
алгоритма, которая возвращает
# стратифицированные папки с данными: каждый набор в папке содержит примерно
такой же процент выборок каждого целевого класса,
# что и полный набор.
skf = StratifiedKFold(n_splits=config.n_folds)
i = 0
for train_split, val_split in skf.split(X_train, train.label_idx):
    K.clear_session()
    X, y, X_val, y_val = X_train[train_split], y_train[train_split],
X_train[val_split], y_train[val_split]
    # В ходе обучения сохраняем веса лучшей модели для потенциального дальнейшего
использования
    checkpoint = ModelCheckpoint('best_%d.h5'%i, monitor='val_loss', verbose=1,
save_best_only=True)
    early = EarlyStopping(monitor="val_loss", mode="min", patience=5)
    tb = TensorBoard(log_dir='.logs\\' + PREDICTION_FOLDER + '\\fold_%i'%i,
write_graph=True)
    callbacks_list = [checkpoint, early, tb]
    print("#"*50)
    print("Fold: ", i)
    model = get_2d_conv_model(config)
    history = model.fit(X, y, validation_data=(X_val, y_val),
callbacks=callbacks_list,
```

```
                      batch_size=64, epochs=config.max_epochs)
    model.load_weights('best_%d.h5'%i)

    # Сохраняем предсказания модели по тренировочным данным
    predictions = model.predict(X_train, batch_size=64, verbose=1)
    np.save(PREDICTION_FOLDER + "\\train_predictions_%d.npy"%i, predictions)

    # Сохраняем предсказания модели по тестовым данным
    predictions = model.predict(X_test, batch_size=64, verbose=1)
    np.save(PREDICTION_FOLDER + "\\test_predictions_%d.npy"%i, predictions)

    # Создание файла с результатами (submission)
    top_3 = np.array(LABELS)[np.argsort(-predictions, axis=1)[:, :3]]
    predicted_labels = [' '.join(list(x)) for x in top_3]
    test['label'] = predicted_labels
    test[['label']].to_csv(PREDICTION_FOLDER + "\\predictions_%d.csv"%i)
    i += 1
```

Результаты проверки работы на платформе Kaggle

Ниже представлен скриншот с результатами проверки работы на платформе Kaggle.

Рисунок 3 – Скриншот результатов из Kaggle

ВЫВОДЫ

В данной лабораторной работе на языке Python (с использованием библиотек Pandas, Scikit-learn, Librosa, и Keras) была создана и настроена модель искусственной нейронной сети для решения задачи классификации произвольных звуков из набора данных «Freesound». Модель свёрточной нейронной сети с использованием слоёв Convolution2D и MaxPooling2D (используется функция активации ReLU для промежуточных слоёв, последний же слой использует функцию softmax), а также классификационных признаков MFCC, получен результат точности 0.78509 (private score) и 0.83333 (public score) после проверки на платформе Kaggle.