Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

Исследование применимости разделяемой по глубине свертки в задачах обработки аудио.

Студент Д.В. Трухан

Руководитель А.В. Леченко

Минск 2021

Введение

Нейронные сети уже давно заняли крепкие позиции в нашей жизни. Машинное обучение присутствует практически во всех сферах жизни современного человека. Нейронные сети помогают сделать наш пользовательский опыт лучше, удобнее. Без них наш цифровой мир было бы тяжело узнать, хотя мы и не задумываемся об этом.

Нейросети это простой в использовании, но очень мощный инструмент. Область применения этого инструмента сложно переоценить. То как и где используются нейросети это лишь небольшая часть их возможности и потенциала.

Меня уже давно интересовал звук. Правда это была больше продюсерская часть чем техническа, но все же вопрос, на чем основана классификация музыки для рекомендаций в популярных стриминговых сервисах, меня не оставил равнодушным.

В этой статье будет рассмотрена тема классификации музыки по жанрам. Также будут рассмотрены сопутствующие классификации темы, такие как спектрограмма и разделяемая по глубине свертка.

Датасет

Мною был взят датасет GTZAN[2]. Данный датасет включает 10 жанров музыки по 100 аудио файлов продолжительностью 30 секунд в каждом.

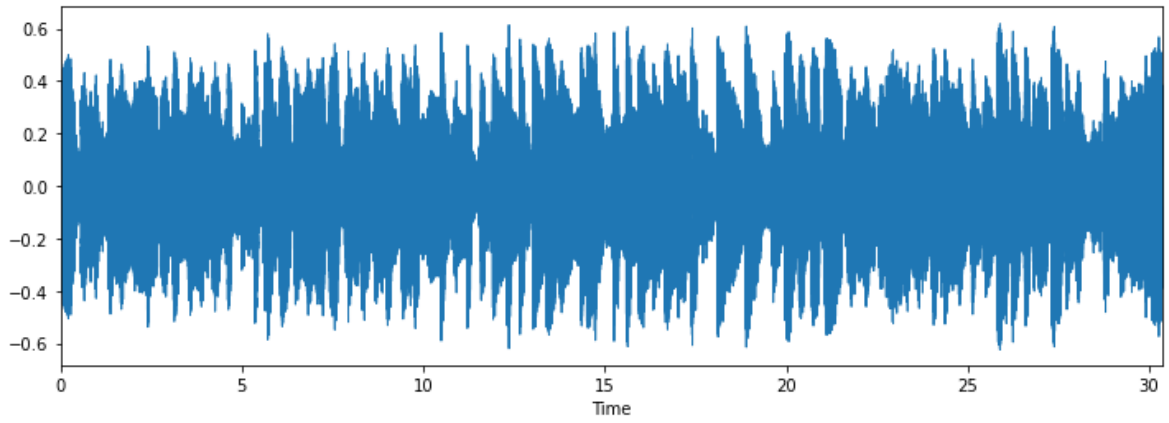
Жанры, включенные в датасет:

1. блюз
2. классика
3. кантри
4. диско
5. хип-хоп
6. джаз
7. металл
8. поп
9. регги
10. рок

Звук

Звук — физическое явление, представляющее собой распространение в виде упругих волн механических колебаний. Ключевое слово здесь волна, поэтому самый очевидный способ представления звука - waveform (звуковая дорожка).

```
data, sr = librosa.load(path)
plt.figure(figsize=(12, 4))
p = librosa.display.waveplot(data, sr=sr)
plt.show()
```

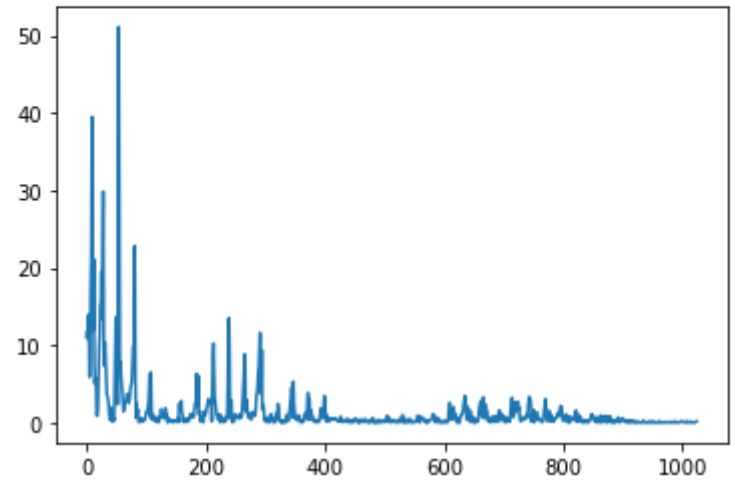


звуковая дорожка 30 секундного отрывка музыки

Однако, такая звуковая дорожка навряд-ли может дать хоть какое-то отдаленное представление о жанре представленной на ней музыки. Waveform может помочь нам понять темпоральную составляющую композиции, а также силу сигнала.

Итак, нам нужно больше информации. Здесь к нам приходит на помощь преобразование Фурье, которое может разложить нашу сложную волну на простейшие синусоиды. Мы переходим из временной области в область представления частотную.

```
stft = librosa.stft(data, hop_length = len(data)+1)
plt.plot(abs(stft))
plt.show()
```

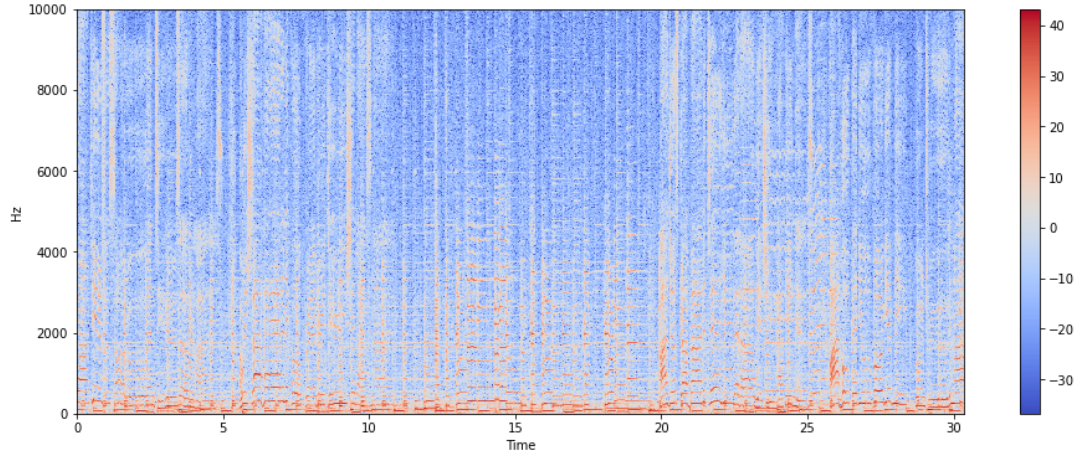


Так, с этим уже можно работать! Но мы потеряли информацию о времени, взамен на частотный слепок всего временного отрезка. Что для нашей задачи окажется большой проблемой, потому что мы воспринимаем музыку во времени. А с преобразованием Фурье по всему временному отрезку мы теряем эту чрезвычайно важную информацию.

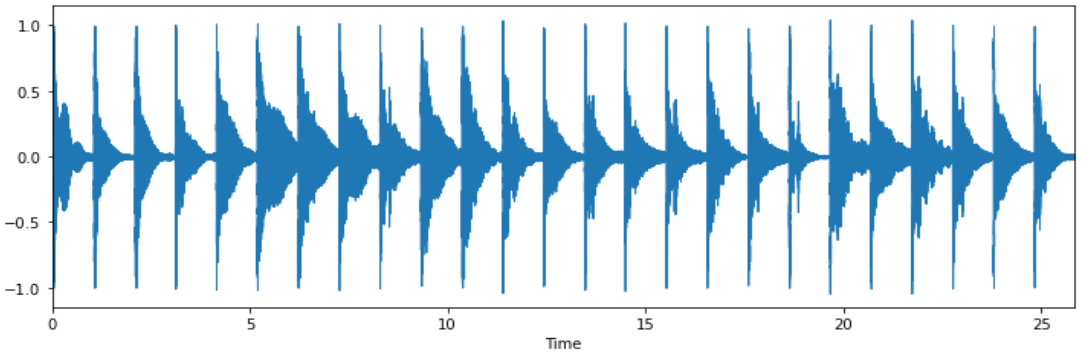
Решение проблемы потери информации о времени является оконное преобразование Фурье (STFT). STFT вычисляет преобразование Фурье по интервалам. Вычисляя Фурье по интервалам, которые сами можем регулировать, мы сохраняем сжатую информацию о времени.

Итак, мы имеем уже 3 измерения. Время, частота и величина(сила). Построив график, где величина будет обозначаться цветом, получим спектрограмму.

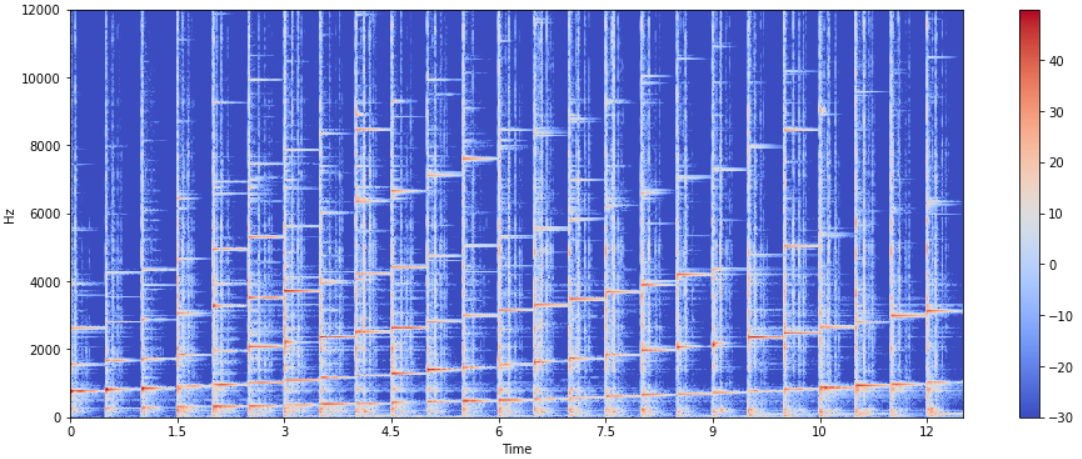
```
stft = librosa.stft(data)
stft_db = librosa.amplitude_to_db(abs(stft))
plt.figure(figsize=(16, 6))
librosa.display.specshow(stft_db, sr=sr, x_axis='time', y_axis='hz')
plt.ylim(0, 10000)
plt.colorbar()
plt.show()
```



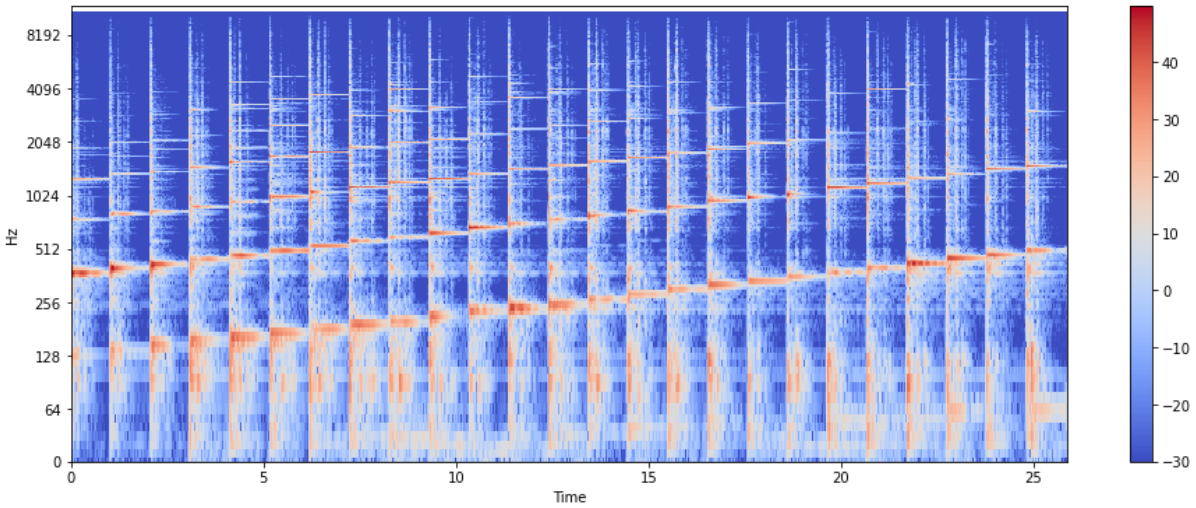
У нас есть отрывок, где с помощью пианино воспроизводится последовательно несколько нот.



И вот спектрограмма.

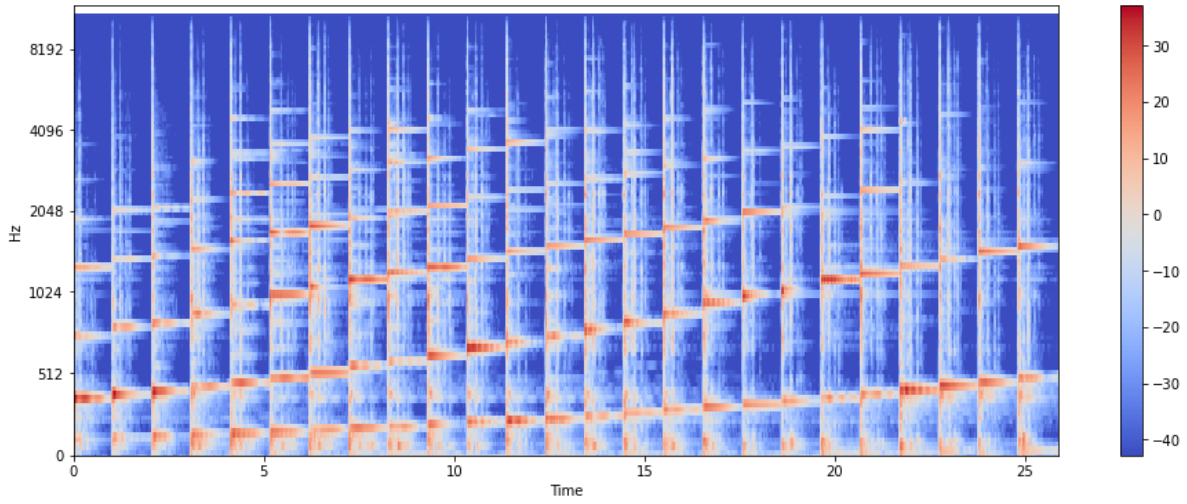


По ней видно, что частоты со временем растут по подобию логарифма. Но на слух то мы ощущаем как тон равномерно увеличивается со временем. И правда, наше ухо устроено так, что на маленьких частотах мы слышим разницу отчетливее, чем на высоких. Разница 500 гц и 1000 слышна гораздо отчетливее чем разница 10000 и 10500. По этой причине будет разумно применять логарифмическую шкалу.



Отлично, теперь по данной спектрограмме мы можем интуитивно понять, что пианист последовательно нажимал на клавиши. Так мы пришли к мел шакале. Мел - единица высоты звука, основанная на субъективном восприятии звука

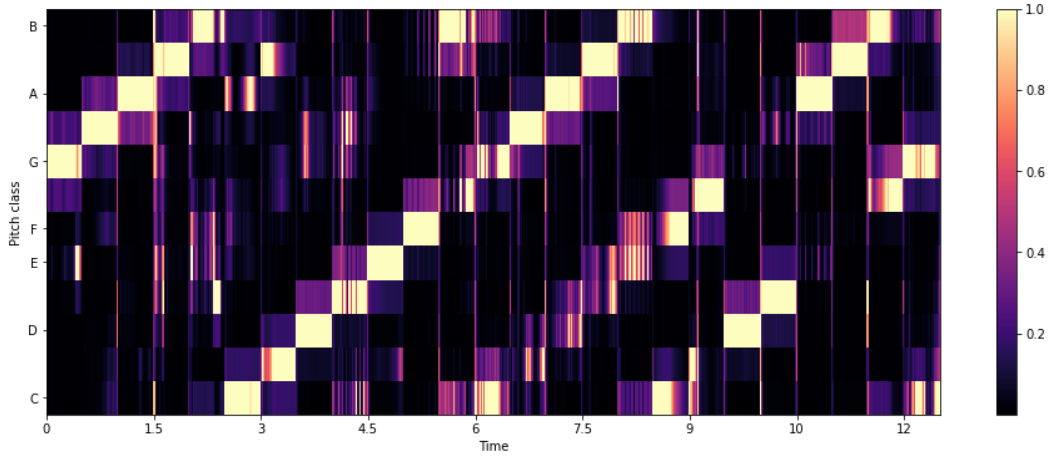
```
stft = librosa.stft(piano)
plt.figure(figsize=(16, 6))
S = librosa.feature.melspectrogram(piano, sr=sr, n_mels=128)
S_dB = librosa.power_to_db(abs(S))
librosa.display.specshow(S_dB, sr=sr, y_axis='mel', x_axis='time')
plt.ylim(0, 12000)
plt.colorbar()
plt.show()
```



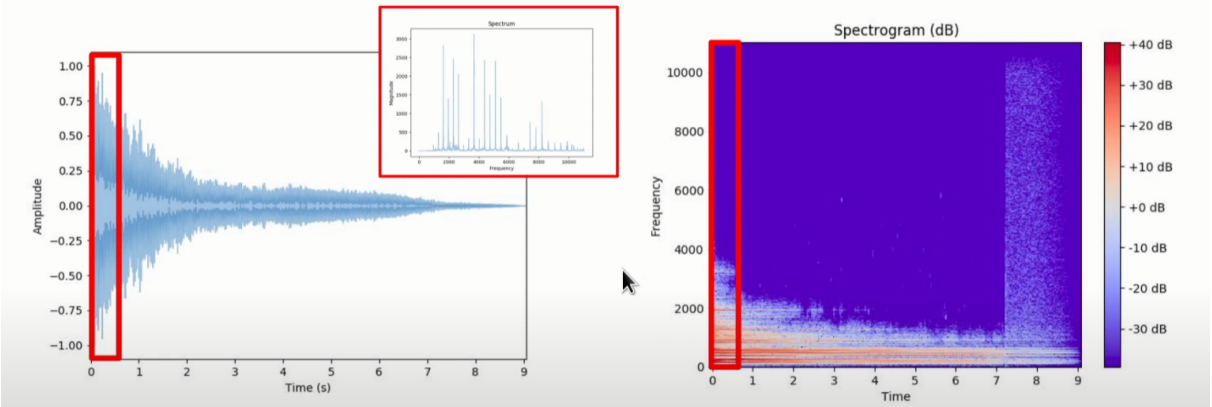
При построении спектрограммы с мел шкалой шкала герц разделяется на участки, которые соответственно переводятся в мел шкалу, используя перекрывающиеся треугольные фильтры.

Сейчас немного о хроматограммах. Применяем банк фильтров, который проецирует звуковую энергию нашей спектрограммы в 12 ячеек, которые соответствуют нотам. Там мы получаем еще один инструмент, характеризующий высоту звука в момент времени, который обширно применяется в обработке композиций. К примеру мы можем взять две хроматограммы одной композиции, исполненной разными музыкантами. Сравнив эти две метрики, может понять, что это и правда одна композиция. Это сравнение, как вариант, может происходить с помощью матрицы расстояний.

```
chroma = librosa.feature.chroma_stft(piano, sr=sr)
plt.figure(figsize=(16, 6))
librosa.display.specshow(chroma, sr=sr,  y_axis='chroma', x_axis='time')
plt.colorbar()
plt.show()
```



Общую картину, преобразования звуковой дорожки в спектрограмму, визуально можно представить так.



Обработка данных

Так как мы не можем работать напрямую с сырыми аудиоданными, по причине их тяжеловесности, приходится искать обходные пути, как-то сжимать, обрабатывать датасет. Из тяжеловесности сети, вследствие огромного входного вектора, следует быстрое переобучение сети.

В статье [2] приведены два способа предварительной обработки данных. Один из способов обработки представляет собой спектрограмму с мел шкалой (melspectrogram). Другой же (mfcc) - это та же спектрограмма с мел шкалой, только разбитая по частотам на 13 ячеек. Для сжатия melspectrogram в mfcc используется дискретное косинусное преобразование, как и в сжатии большинства форматов изображений, например jpeg. До сжатия данные предварительно приводятся к логарифмической шкале.

Для увеличения набора данных можно разделять 30 секундные отрывки музыки на сегменты.

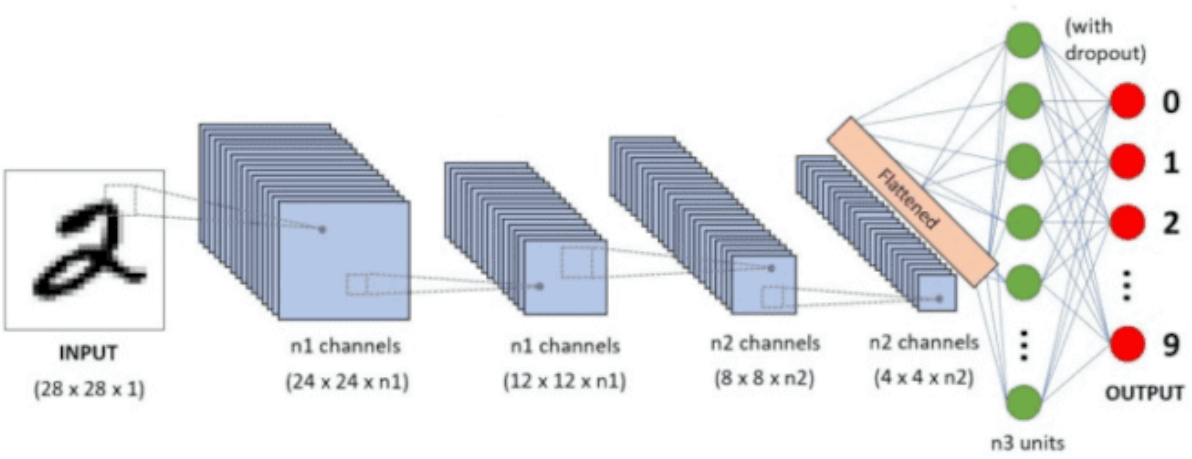
Сверточные нейронные сети

Нейросеть представляет собой последовательность нейронов, соединенных между собой. Такая простая, но гениальная идея пришла в мир программирования из биологии. Нейронная сеть это попытка оцифровки структуры человеческого мозга. Этот инструмент активно используется для решения множества задач, самыми распространенными из которых являются: предсказание, распознавание и классификация. Последнее и является задачей этой статьи.

Одно из главных преимуществ нейросети — это ее обучаемость. В отличие от обычных, всем привычных алгоритмов, которые пишет человек, нейронная сеть способна сама подобрать нужную функцию, алгоритм, путем обучения. Обучение заключается в том, что связи между нейронами регулируются, путем изменения весовых коэффициентов.

Как известно наука очень много берет из природы, и тут она не остановилась. Сверточная нейронная сеть - специальная архитектура нейронной сети, предназначенная для эффективного распознавания изображений. Идея сверточной нейронной сети заключается в чередовании сверточных слоев (convolution layers) и субдискретизирующих слоев (pooling layers).

Сверточные сети стали моделью реальной зрительной коры. Все это дает возможность распознавания изображений, путем перехода от конкретных деталей, например линии, к более и более абстрактным деталям, понятиям. Но практике не бывает такого, что, после обучения сети распознавать лица, отобразив свертку, мы увидим там части лица. В основном параметры сверточный сети не несут никакого смысла для человека.



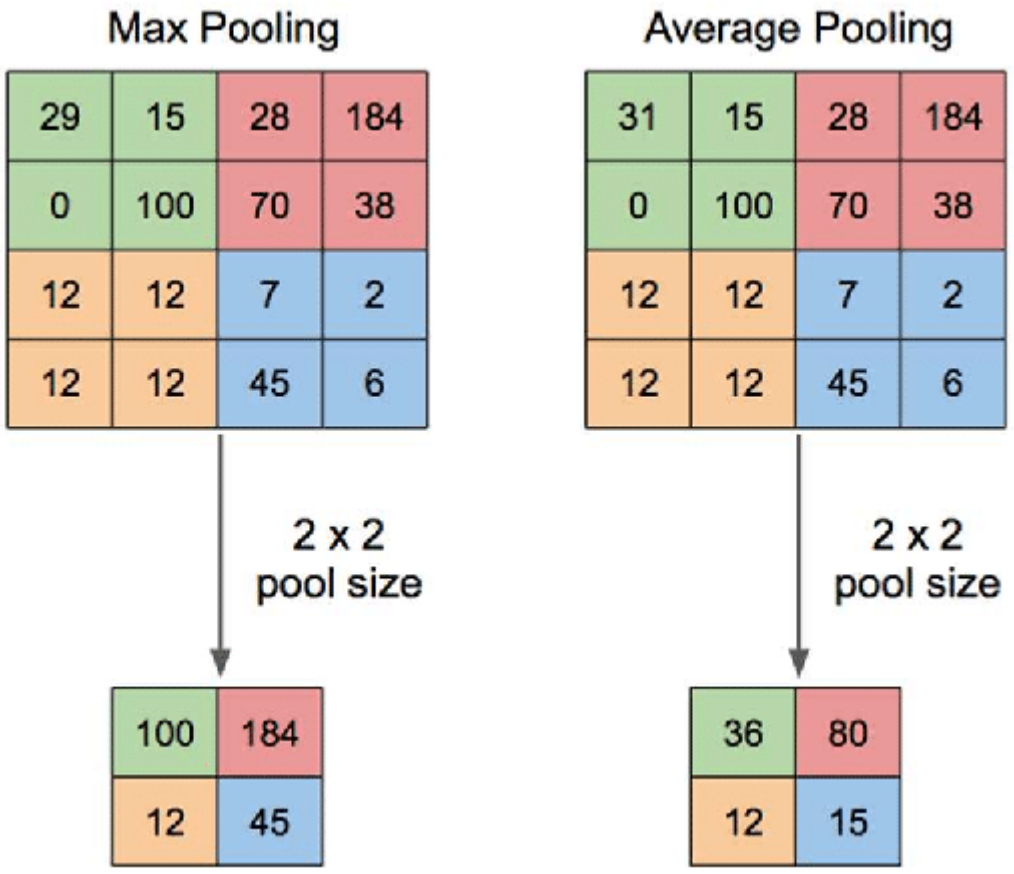
В сверточной нейронной сети используется матрица весов, пройдя которой по всему изображению, если слой является входным, либо по карте признаков, полученной ранее, получаем новое изображение (карту признаков). Такая матрица с весовыми коэффициентами называется ядром свертки. Карта признаков получается путем последовательного применения ядра свертки ко всему изображению. Количество ядера, соответсвенно и карт признаков, задается архитектурой сверточной сети. Эти ядра подвергаются обучению, т.е. подвержены алгоритму обратного распространения ошибки (backpropagation), а не закладываются изначально, как можно было подумать.

Преимущество сверточной сети это небольшое количество параметров. Если подать изображение целиком на вход

полносвязной сети, размер входного слоя этой сети должен быть равен количеству пикселей изображение. Также сверточная сеть помогает предотвратить быстрое переобучение, по сравнению с полносвязной сетью.

По итогу мы имеем изображение, ядра, с помощью которых происходит свертка данного изображения, и на выходе мы получаем карты признаков. В процессе свертки изображения кроме ядра свертки появляются дополнительные параметры такие, как шаг прохода свертки по каналу и отступы по краям канала. Данные параметры влияют как на качество обучения, так и на размер выходных каналов и задаются архитектором нейронной сети.

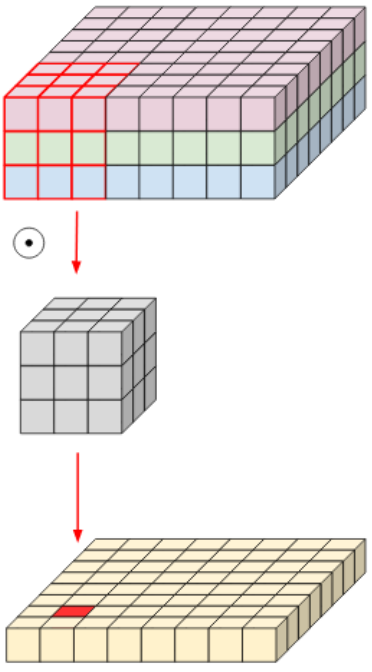
Субдискретизация (pooling) - операция уменьшения размерности карт признаков сформированных сверточным ядром. Очень грубо говоря это уменьшение качества картинки. Из нескольких рядом стоящих пикселей выбирается один по какому-либо критерию, например максимальный элемент. Уплотнение карты признаков позволяет уменьшить количество параметров, что увеличивает скорость обучения и частично предотвращает переобучение.



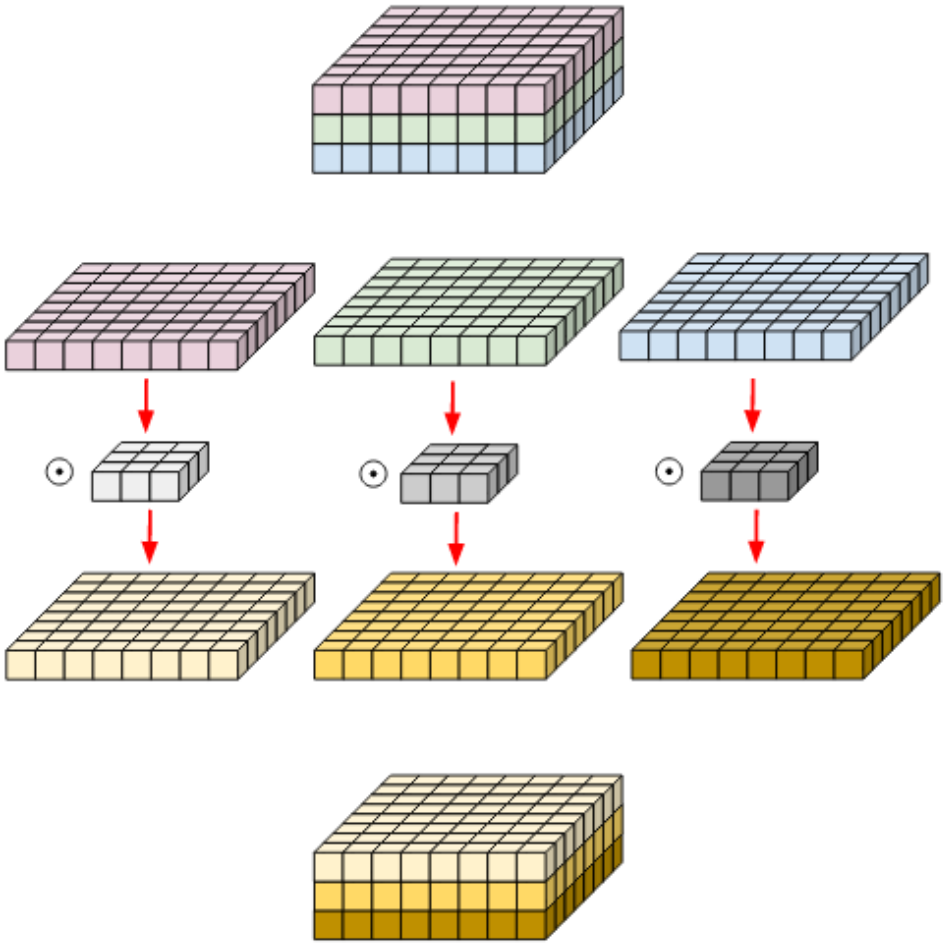
По итогу типичная сверточная сеть состоит из нескольких слоев, а иногда и большого их количества. Изображение проходит через свертку, далее полученные каналы следуют на следующий слой, называемый слоем субдискретизации, после которого на выход подаются уплотненные карты признаков. После череды сверточных слоев и слоев субдискретизации карта признаков либо сама собой превращается в вектор, либо ее намеренно трансформируют в вектор, чтобы подать на следующий слой. Этим следующим слоем является полносвязная нейронная сеть, которая преобразует вектор карт признаков в необходимый нам выход.

Разделяемые по глубине свертки

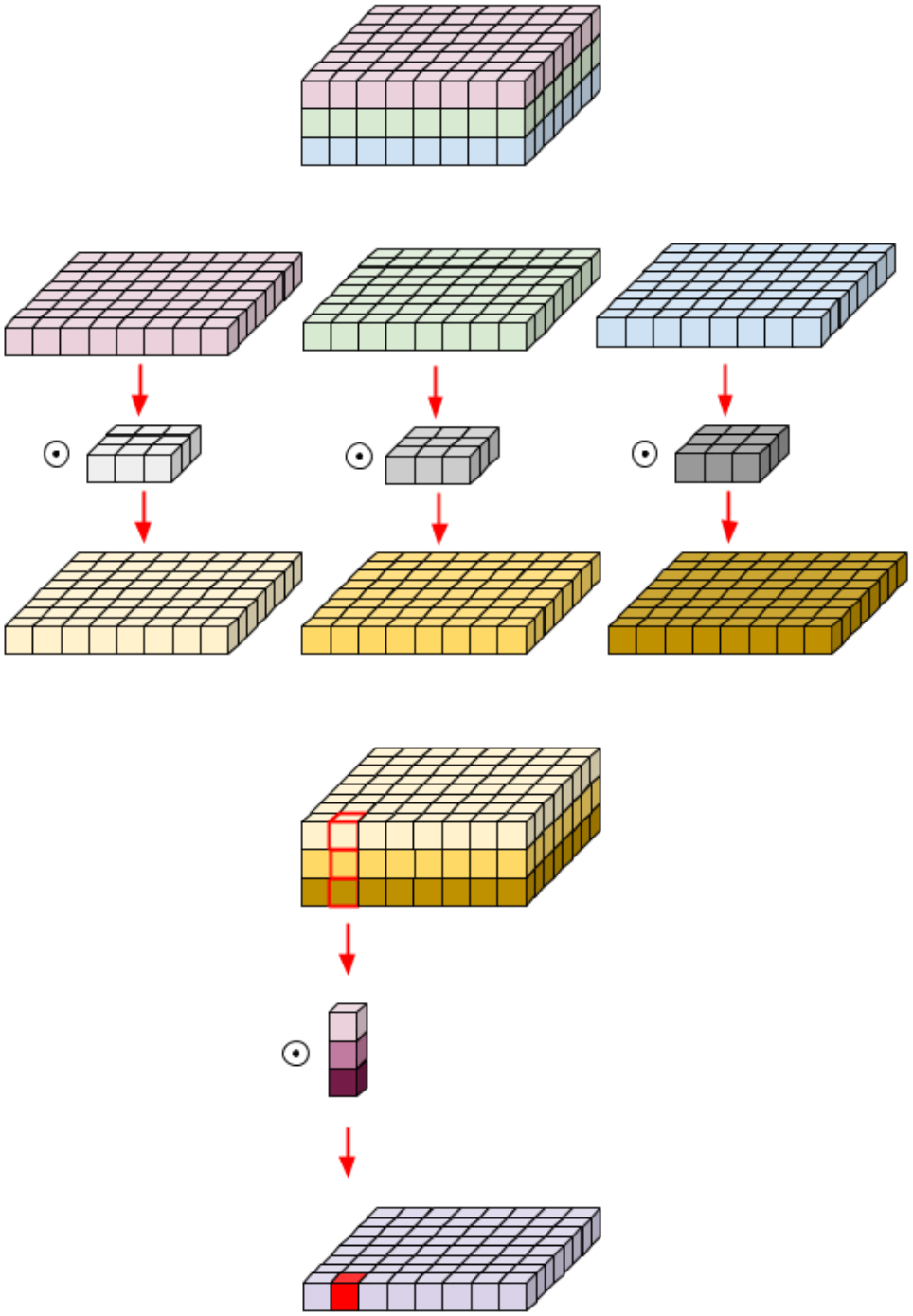
Рассмотрим в начале стандартные свертки. Такая свертка проходит по всем каналам одновременно, на выходе получается один канал. Т.е. для получения одного выходного канала нам потребуется (ширина)*(высота)*(кол-во входных каналов) параметров. Если мы хотим получить несколько выходных каналов, нам потребуется еще умножить количество параметров на желаемое количество выходных каналов. Итого мы получаем внушительное количество обучаемых параметров. Потому что для каждого выходного канала мы используем отдельный фильтр.



В данной свертке одним этапом каналы и фильтруются и объединяются. А что если разнести эти две операции? Будем использовать отдельный фильтр на каждый канал. На выходе мы будем получать столько же каналов как и на входе.



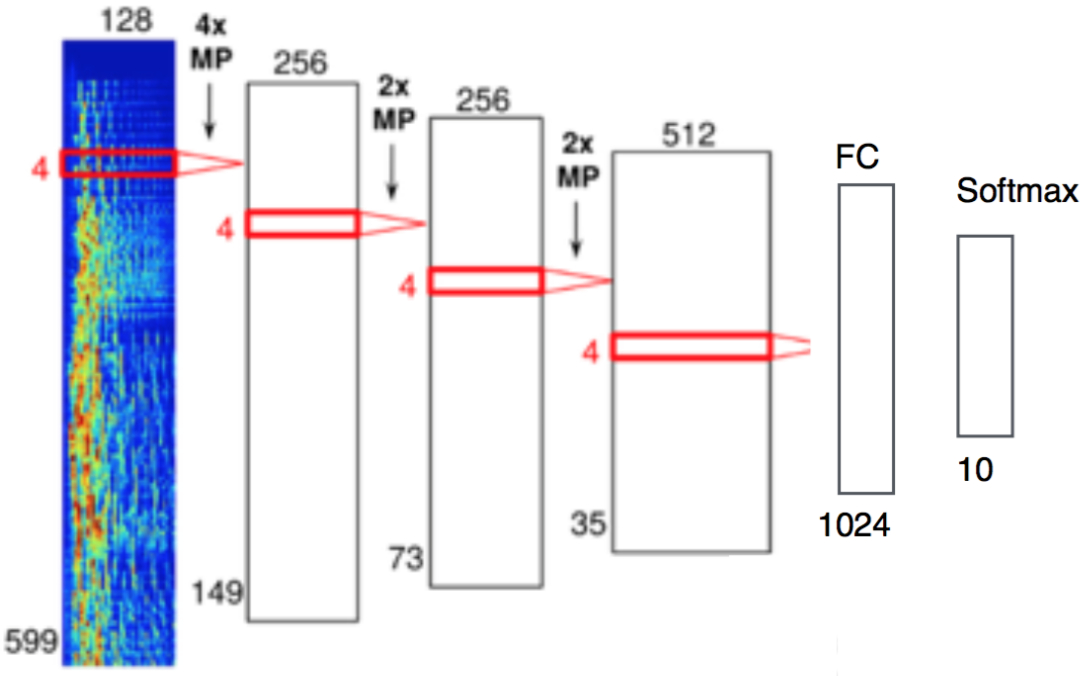
На данном этапе мы получаем отфильтрованные каналы. Теперь из них мы можем порождать нужное нам количество каналов при помощи фильтра 1*1.



В разделяемой по глубине свертке, в отличие от стандартной свертки, каналы порождаются фильтром 1*1, что гораздо менее затратнее по параметрам. В общем случае разделяемые по глубине свертки ускоряют обучение, из-за меньшего количества параметров, а также уменьшают вероятность переобучения. Однако, при таком уменьшении емкости модели, модель может перестать обучаться до требуемого уровня.

Архитектура сети

В статье [1] используется стандартная сверточная сеть с 3 сверточными слоями с ядром 4*4, чередующимися с max pool слоями для уменьшения размерности каналов. Также используется dropout слой для борьбы с переобучением. В конце идет полносвязная сеть с одним скрытым слоем и 10 выходами. В скрытом слое 2^13 параметров.



Представлена картинка используется для классификации музыки обработанной в мел спектрограммы. В случае с MFCC входная размерность будет 13 вместо 128.

Во всех обучениях использовался оптимизатор Adam со скоростью обучения 0.0001. Функцией потерь была выбрана категориальная кросс энтропия. Размер партии (batch size) определился как 64.

```
model.compile(optimizer=keras.optimizers.Adam(
                          learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.summary()

history = model.fit(x_train, y_train,
                    validation_data=(x_val, y_val),
                    batch_size=64, epochs=500)

test_loss, test_acc = model.evaluate(x_test, y_test,
                                     verbose=2)
print('\ntest:', test_acc)
```

Выборка данных делилась на тренировочную, валидационную и проверочную.

```
x_train, x_val, x_test, y_train, y_val, y_test =
prepare_dataset(x, y, 0.25, 0.2)
```

Графики точности и потерь рисовались с помощью библиотеки matplotlib.

Изначальную модель я представил в таком виде.

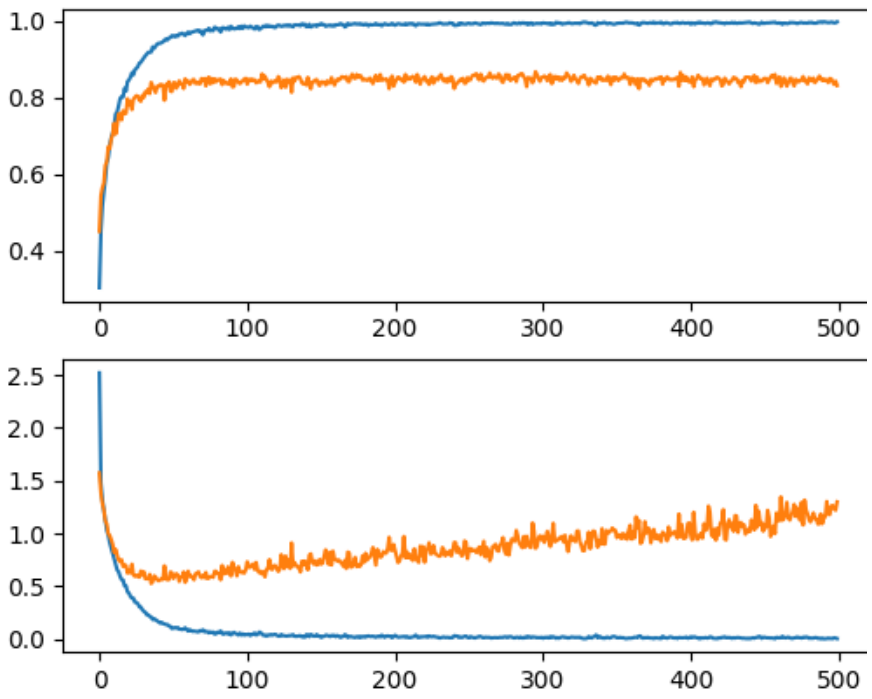
```
model = keras.Sequential([
    Conv2D(256, (4, 13), activation='relu',
    input_shape=input_shape),
    MaxPooling2D((4, 1), 4),
    Dropout(0.3),

    Conv2D(256, (4, 1), activation='relu'),
    MaxPooling2D((2, 1), 2),
    Dropout(0.3),

    Conv2D(512, (4, 1), activation='relu'),
    MaxPooling2D((2, 1), 2),
    Dropout(0.3),

    Flatten(),
    Dense(2 ** 13, activation='relu'),
    Dropout(0.3),

    Dense(10, activation='softmax'),
])
```



Данная модель показывает неплохой результат. Обучение эпохи происходит около 3 секунд. За 500 эпох модель может похвастаться точностью 83-87% на тестовой выборке.

Попробуем заменить сертки на долгожданные разделяемые по глубине.

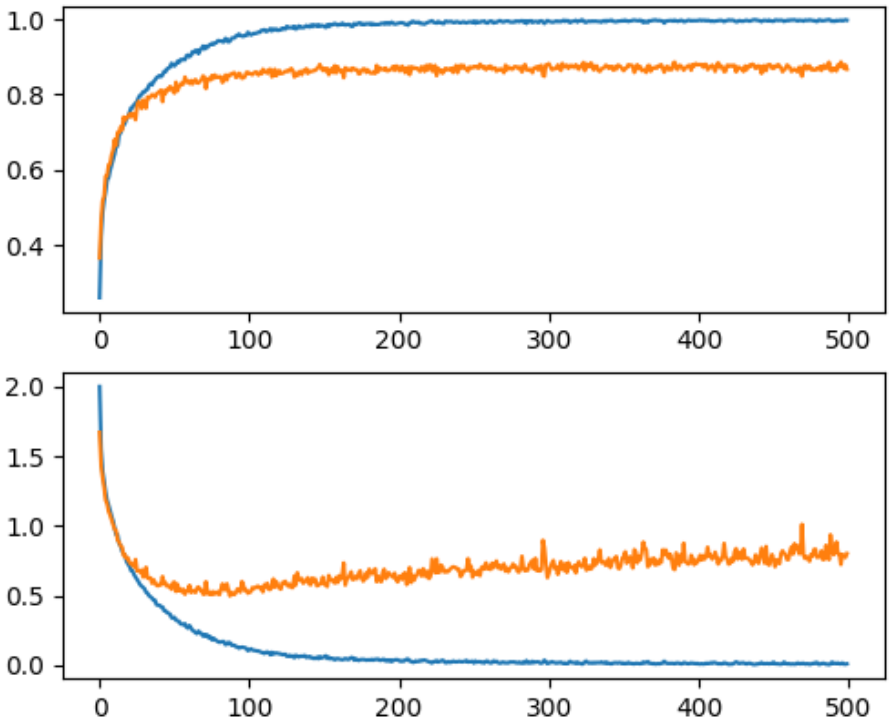
```
model = keras.Sequential([
    Conv2D(256, (4, 13), activation='relu',
    input_shape=input_shape),
    MaxPooling2D((4, 1), 4),
    Dropout(0.3),

    SeparableConv2D(256, (4, 1), activation='relu'),
    MaxPooling2D((2, 1), 2),
    Dropout(0.3),

    SeparableConv2D(512, (4, 1), activation='relu'),
    MaxPooling2D((2, 1), 2),
    Dropout(0.3),

    Flatten(),
    Dense(2 ** 13, activation='relu'),
    Dropout(0.3),

    Dense(10, activation='softmax'),
])
```



Как ни странно при таком сетапе точность даже выросла и составила 86-88%. Время обучения эпохи незначительно сократилось. Такая модель на старте немного медленнее обучается, но показывает более высокую стабильность, меньше переобучается.

Далее я пытался всячески улучшить эту модель, варьируя параметры. Добавил batch normalization, регуляризацию L2. Все это лишь немного увеличило точность и стабильность.

Следующим шагом я начал тестировать более глубокую архитектуру. Эту архитектуру я подсмотрел у энкодера UnetX, но адаптировал под свою задачу. Сверткам лучше двигаться только по временной оси, что ускоряет обучение.

```
model = keras.Sequential([
    Conv2D(256, (4, 13), activation='relu',
    kernel_regularizer=keras.regularizers.l2(0.04),
    input_shape=input_shape),
    BatchNormalization(),
    Dropout(0.2),
    SeparableConv2D(256, (4, 1), activation='relu',
    kernel_regularizer=keras.regularizers.l2(0.04)),
    BatchNormalization(),
```

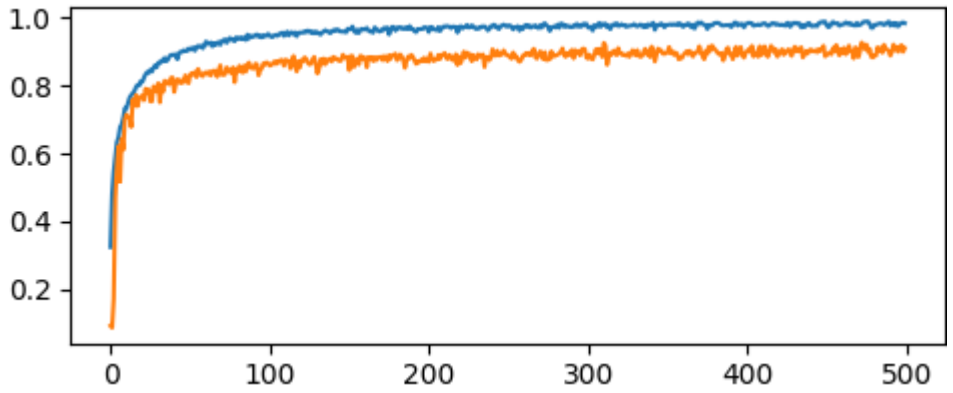
```
    MaxPooling2D((2, 1), 2),

    SeparableConv2D(128, (3, 1), activation='relu'),
    BatchNormalization(),
    Dropout(0.2),
    SeparableConv2D(128, (3, 1), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 1), 2),

    SeparableConv2D(64, (2, 1), activation='relu',
    kernel_regularizer=keras.regularizers.l2(0.04)),
    Dropout(0.2),
    BatchNormalization(),
    SeparableConv2D(64, (2, 1), activation='relu',
    kernel_regularizer=keras.regularizers.l2(0.04)),
    BatchNormalization(),
    MaxPooling2D((2, 1), 2),

    Flatten(),
    Dense(2 ** 10, activation='relu',
    kernel_regularizer=keras.regularizers.l2(0.08)),
    Dropout(0.2),

    Dense(10, activation='softmax'),
])
```



Такой сетап смог достичь точности 90-94% за 500 эпох. Скорость прохода эпохи все та же, около 3 секунд. Данная модель еще лучше борется с переобучением.

	Изначальная модель	Модель с разделяемыми по глубине свертками	Более глубокая модель с разделяемыми по глубине свертками
Время предсказания тестовой выборки, сек	0.435	0.412	0.573

Вывод

Технологии по распознаванию изображений на практике применимы к отработанному звуку. Мел спектрограммы и мел-частотный кепстр хорошо подходят для обработки звука перед обучением нейронной сети.

Основные силы в этой статье были брошены на более легковесный мел-частотный кепстр (около 100 обучений). В будущем нужно попробовать подобрать параметры для обучения сети на мел спектрограммах. Вероятно, с мел спетрограммами можно добится точности выше.

Список используемых источников

1. https://github.com/mlachmish/MusicGenreClassification/blob/master/Music%20Genre%20Classification%20Using%20Deep%20Learning.pdf
2. https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification
3. https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec
4. https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53
5. https://musicinformationretrieval.com/spectral_features.html
6. https://benanne.github.io/2014/08/05/spotify-cnns.html
7. https://libeldoc.bsuir.by/bitstream/123456789/45774/1/Adaska_UNetX.pdf