## Задание 2.

В обоих задачах применялась библиотека глубокого обучения PyTorch.

### Классификация.

Для решения задачи классификации использовалась глубокая свёрточная нейронная сеть с архитектурой, похожей на популярную сеть VGGNet (<https://arxiv.org/abs/1409.1556>).

**Вход**

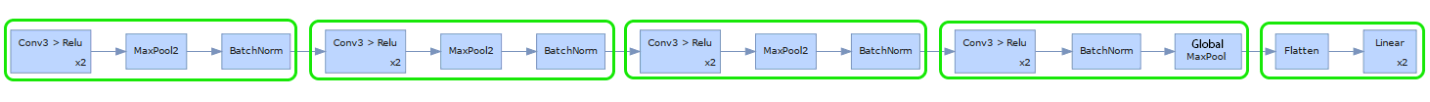
На вход сеть принимает mel-спектрограммы любой длины *spectrogram\_length* в виде массива с размерностями [*spectrogram\_length*, 80], которые рассматриваются в качестве одномерных сигналов с количеством каналов, равным 80.

Для обучения с учителем каждой mel-спектрограмме соответствует метка класса: зашумлённый элемент - класс 1, чистый элемент – класс 0.

**Выход**

Данная сеть на выходе возвращает вещественное число, но не обязательно из промежутка (0, 1). Предполагается, что приведение выхода к виду вероятности из интервала (0, 1) осуществляется вне обучения с помощью сигмоидальной функции, что обеспечивает численно более устойчивый процесс обучения.

**Архитектура**

Визуально архитектура представлена на рисунке ниже и условно может быть разделена на 5 стадий:

* 1-3 стадии состоят из блоков одинаковой структуры: 2 одномерных свёрточных слоя с ядром 3, не изменяющих пространственного размера, и нелинейностями ReLU, за которыми следуют слои Max-pooling с ядром 2 и шагом 2, и слой Batch-normilization.
* 4ая стадия отличается тем порядком применения слоёв Max-pooling и Batch-normilization, причём в качестве Max-pooling используется так называемый Global max-pooling, размером ядра которого является вся карта признаков. Global max-pooling в данном случае обеспечивает возможность сети принимать на вход данные *любой длины*.
* 5ая стадия представляет собой формальны слой Flatten вытягивания данных в одномерный массив и 2 полносвязных слоя Linear с количеством нейронов 10 и 1 соответственно. Причём после первого слоя Linear применяется нелинейность ReLU, а второй слоя Linear считается выходом сети.

**Функция потерь**

В качестве функции потерь использовалась бинарная кросс-энтропия, которая предварительно применяет ко входу сигмоидальную функцию.

**Алгоритм оптимизации**

В качестве алгоритма минимизации функции потерь использовалась модификация алгоритма градиентного спуска Adam.

**Процесс обучения**

Для обучения использовались данные из папки **train1**, для финальной оценки качества – данные из папки **val**.

Во время обучения сеть на вход принимала данные порциями (батчами). Формирование батча происходило следующим образом: из обучающей выборки случайно выбиралась группа из 64х элементов, после чего вычислялась длина минимального среди них элемента. Все элементы приводились к одинаковому размеру путём случайной «обрезки» до вычисленной длины.

Для контроля обучения данные **train1** случайным образом разделены на две подвыборки в соотношении 9 к 1: **train** и **dev**. Выборка **dev** использовалась для реализации подхода *early stopping* (ранней остановки), направленной на предотвращение переобучения модели на подвыборке **train**, а в качестве контрольной метрики рассматривалась целевая метрика *accuracy*: если *accuracy* в течение 3х эпох обучения на подвыборке **dev** не увеличивается, то процесс обучения останавливается.

**Оценка результата**

Для оценки качества модели выбиралась модель, лучшая с точки зрения метрики accuracy на подвыборке **dev**, после чего тестировалась на всей выборке **val** (на вход сети элементы подавались по одному, каждый в своём исходном размере).

**Результат на данных val**

**accuracy = 98.3%**

**Классификатор**

**Код для обучения**

Весь код для обучения содержится в формате *jupyter notebook* в файле с названием **goznak\_junior\_nbk.ipynb**

**Код для тестирования**

Код для тестирования содержится в формате *jupyter notebook* в файле с названием **goznak\_junior\_nbk\_test.ipynb**

Для успешного запуска необходимо в папке с файлом **goznak\_junior\_nbk\_test.ipynb** наличие файла с весами для сети **classifier\_epoch15.pth**.

Для тестирования необходимо в ячейке с названием «**Path to folder with test data**» задать значение переменной **test\_path** относительный или абсолютный путь до папки с данными для тестирования. По данному пути должна находиться папка, содержащая структуру, как и данные для обучения: включающую папки «**noisy»** и «**clean»** с данными, расположенными в подпапках**.**

Далее необходимо выполнить действия во всех ячейках, тогда в последней ячейке раздела «**Classifier evaluation»** будет выведен результат.

### Устранение шума.

Для решения задачи устранения шума использовалась глубокая свёрточная нейронная сеть с архитектурой, похожей на популярную сеть DnCNN (<https://arxiv.org/abs/1608.03981>).

Пусть X – зашумлённый сигнал, Y – чистый сигнал, а N – шум, наложенный на Y. Из предположения, что X = Y + N, сеть учится вычислять шумовой остаток N, а не чистый сигнал Y напрямую.

**Вход**

На вход сеть принимает mel-спектрограммы любой длины *spectrogram\_length* в виде массива с размерностями [*spectrogram\_length*, 80], которые рассматриваются в качестве одномерных сигналов с количеством каналов, равным 80.

Для обучения с учителем каждой зашумлённой mel-спектрограмме X соответствует её шумовой остаток, вычисленный как N=X – Y.

**Выход**

Данная сеть на выходе возвращает предсказанный шумовой остаток N. Делается это из тех соображений, что получение чистого сигнала из зашумлённого не сильно отличается от единичного оператора, а сети сложнее обучаться для такой задачи.

**Архитектура**

Сеть состоит из 15ти свёрточных слоёв. Архитектура довольно проста и условно может быть разделена на 3 части:

* Начальная. Состоит из одномерного свёрточного слоя с ядром 3, не изменяющего пространственного размера, и нелинейности ReLU.
* Средняя. Состоит из 13ти повторяющихся блоков одинаковой структуры: одномерного свёрточного слоя с ядром 3, не изменяющего пространственного размера, слоя Batch-normalization и нелинейности ReLU.
* Финальная. Состоит из одного одномерного свёрточного слоя с ядром 3, не изменяющего пространственного размера, без нелинейности.

Также для регуляризации дополнительно после каждого четвёртого блока со свёрточным слоем использовался слой Dropout, который во время обучения обнулял половину значений выходов слоёв сети, после которых он применялся.

**Функция потерь**

В качестве функции потерь использовалась метрика среднеквадратичной ошибки MSE loss.

**Алгоритм оптимизации**

В качестве алгоритма минимизации функции потерь использовалась модификация алгоритма градиентного спуска Adam.

**Процесс обучения**

Для обучения использовались данные из папки **train1**, для финальной оценки качества – данные из папки **val**.

Во время обучения сеть на вход принимала данные порциями (батчами). Формирование батча происходило следующим образом: из обучающей выборки случайно выбиралась группа из 64х элементов, после чего вычислялась длина минимального среди них элемента. Все элементы и соответствующие им целевые шумовые остатки синхронно приводились к одинаковому размеру путём случайной «обрезки» до вычисленной длины.

Для контроля обучения данные **train1** случайным образом разделены на две подвыборки в соотношении 9 к 1: **train** и **dev**. Выборка **dev** использовалась для реализации подхода *early stopping* (ранней остановки), направленной на предотвращение переобучения модели на подвыборке **train**, а в качестве контрольной метрики рассматривалась целевая метрика *MSE*: если *MSE* в течение 3х эпох обучения на подвыборке **dev** не увеличивается, то процесс обучения останавливается.

**Оценка результата**

Для оценки качества модели выбиралась модель, лучшая с точки зрения метрики MSE на подвыборке **dev**, после чего тестировалась на всей выборке **val** (на вход сети элементы подавались по одному, каждый в своём исходном размере).

**Результат на данных val**

**MSE = 0.055**

**Устранение шума**

**Код для обучения**

Весь код для обучения содержится в формате *jupyter notebook* в файле с названием **goznak\_middle\_nbk.ipynb**

**Код для тестирования**

Код для тестирования содержится в формате *jupyter notebook* в файле с названием **goznak\_middle\_nbk\_test.ipynb**.

Для успешного запуска необходимо в папке с файлом **goznak\_middle\_nbk\_test.ipynb** наличие файла с весами для сети **denoiser\_epoch4.pth**.

Для тестирования необходимо в ячейке с названием «**Path to folder with test data**» задать значение переменной **test\_path** относительный или абсолютный путь до папки с данными для тестирования. По данному пути должна находиться папка, содержащая структуру, как и данные для обучения: включающую папки «**noisy»** и «**clean»** с данными, расположенными в подпапках**.**

Далее необходимо выполнить действия во всех ячейках, тогда в последней ячейке раздела «**Denoising evaluation»** будет выведен результат.

**\***Опционально можно также задать список порядковых индексов подгруженных данных в переменной **list2visualize**. В таком случае в последней ячейке раздела «**Results visualisation**» будет посчитана метрика MSE для каждой mel-спектрограммы из списка, а также выведены последовательно mel-спектрограммы сперва для зашумлённого сигнала (входа), затем для предсказанного сетью очищенного результата, затем для чистого сигнала.