Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики

ШУМОПОДАВЛЕННЯ З ВИКОРИСТАННЯМ ГЛИБОКОГО НАВЧАННЯ

Текстова частина до курсової роботи за спеціальністю
“Інженерія програмного забезпечення” 121

Керівник курсової роботи кандидат
фізико-математичних наук, доцент
Крюкова Галина Віталіївна

_____
(підпис)
“____”_____2020 р.

Виконав студент
Марченко Віталій Вікторович
“____”_____2020 р.

Київ 2020

Міністерство освіти і науки України

Зміст

Тема: Шумоподавлення з використанням глибокого навчання

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	березень 2020 р.	
2.	Огляд літератури за темою роботи.	березень 2020 р.	
3.	Розробка алгоритму	квітень 2020 р.	
4.	Написання пояснювальної роботи	квітень 2020 р.	
7.	Створення слайдів для доповіді та написання доповіді.	01.04.2020	
8.	Аналіз отриманих результатів керівником.	10.04.2020	
8.	Коригування роботи за результатами аналізу.	15.04.2020	
10.	Остаточне оформлення пояснювальної роботи та слайдів.	17.04.2020	
11.	Захист курсової роботи.	19.04.2020	

Студент **Марченко Віталій Вікторович**

Керівник **Крюкова Галина Віталіївна**

“_____”_____ 2020 р.

Анотація

Ця робота описує підхід до шумоподавлення звукових сигналів, що містять людську мову, з використанням глибокого навчання (deep learning).

Вступ

Мета алгоритму - отримавши на вхід звуковий сигнал, що складається з людської мови та фонових шумів, на виході видати звуковий сигнал тільки з людською мовою. Алгоритм базується на згорточній мережі з агрегацією контекста (convolutional context aggregation network). Loss-функція цієї мережі заснована на порівнянні ваг з вагами іншої мережі, натренованої для розпізнавання звукової середи (acoustic environment detection and domestic audio tagging).

1. Опис алгоритму

1.1 Структура мережі

Мережа складається з 16 згорткових слоїв (convolutional layers). Перший та останній слої є 1-вимірними тензорами розмірності N. Число N може варіюватися, тобто від нас не вимагається знайти його наперед.

Кожен проміжний слой представляє з себе 2-вимірний тензор розмірності NxW, де W - кількість feature maps у кожному з слоїв.

Параметри кожного з проміжних слоїв вираховують базуючись на параметрах попереднього слою, накладаючи матрицю згортки (convolutional kernel) розміру 3x1, а потім leaky rectified linear unit (LReLU) max(0.2x, x).

Loss-функція має наступний вигляд:

$$\mathcal{L}_{ß,\mathbf{x}}(\theta) = \sum_{m=1}^{M} \lambda_m \left\| \Phi^m(ß) - \Phi^m(g(\mathbf{x};\theta)) \right\|_1$$

1.2 Датасет

Для тренування мережі був використаний датасет з публікації [2], адже, серед усіх знайдених мною датасетів, цей містить найбільше різних шумів, а також як чоловічі, так і жіночі голоси, що робить мережу, натреновану на ньому, придатною для використання у найбільшому спектрі ситуацій.

Список використаних джерел

1. Germain, F.G., Chen, Q., Koltun, V. (2019) Speech Denoising with Deep Feature Losses. [Електроний ресурс] - Режим доступу: https://arxiv.org/pdf/1806.10522.pdf
2. C. Valentini-Botinhao, X. Wang, S. Takaki, and J. Yamagishi, "Investigating RNN-based speech enhancement methods for noise-robust text-to-speech," [Електроний ресурс] - Режим доступу: http://ssw9.talp.cat/papers/ssw9_PS2-4_Valentini-Botinhao.pdf
3. Michelashvili, M., Wolf, L. (2019) Audio Denoising with Deep Network Priors. [Електроний ресурс] - Режим доступу: https://www.researchgate.net/publication/332462978_Audio_Denoising_with_Deep_Network_Priors

Додаток А

Функція класифікації

```
    # FEATURE LOSS NETWORK
    def lossnet(input, n_layers=14, training=True, reuse=False,
norm_type="SBN", ksz=3, base_channels=32, blk_channels=5):
        layers = []

        if norm_type == "NM": # ADAPTIVE BATCH NORM
            norm_fn = nm
        elif norm_type == "SBN": # BATCH NORM
            norm_fn = slim.batch_norm
        else: # NO LAYER NORMALIZATION
            norm_fn = None

        for id in range(n_layers):
            n_channels = base_channels * (2 ** (id // blk_channels)) # UPDATE
CHANNEL COUNT

            if id == 0:
                net = slim.conv2d(input, n_channels, [1, ksz], activation_fn=lrelu,
normalizer_fn=norm_fn, stride=[1, 2],
                                  scope='loss_conv_%d' % id, padding='SAME',
reuse=reuse)
                layers.append(net)
            elif id < n_layers - 1:
                net = slim.conv2d(layers[-1], n_channels, [1, ksz], activation_fn=lrelu,
normalizer_fn=norm_fn,
                                  stride=[1, 2], scope='loss_conv_%d' % id,
padding='SAME', reuse=reuse)
                layers.append(net)
            else:
                net = slim.conv2d(layers[-1], n_channels, [1, ksz], activation_fn=lrelu,
normalizer_fn=norm_fn,
                                  scope='loss_conv_%d' % id, padding='SAME',
reuse=reuse)
                layers.append(net)

        return layers
```

Додаток Б

Loss-функція

```
    def featureloss(target, current, loss_weights, loss_layers, n_layers=14,
norm_type="SBN", base_channels=32, blk_channels=5):

        feat_current = lossnet(current, reuse=False, n_layers=n_layers,
norm_type=norm_type,
                               base_channels=base_channels, blk_channels=blk_channels)

        feat_target = lossnet(target, reuse=True, n_layers=n_layers,
norm_type=norm_type,
                              base_channels=base_channels, blk_channels=blk_channels)

        loss_vec = [0]
        for id in range(loss_layers):
            loss_vec.append(l1_loss(feat_current[id], feat_target[id]) /
loss_weights[id])

        for id in range(1,loss_layers+1):
            loss_vec[0] += loss_vec[id]

        return loss_vec
```

Додаток В

Функція шумоподавлення

```
    # ENHANCEMENT NETWORK
    def senet(input, n_layers=13, training=True, reuse=False, norm_type="NM",
              ksz=3, n_channels=32):

        if norm_type == "NM": # ADAPTIVE BATCH NORM
            norm_fn = nm
        elif norm_type == "SBN": # BATCH NORM
            norm_fn = slim.batch_norm
        else: # NO LAYER NORMALIZATION
            norm_fn = None

        for id in range(n_layers):

            if id == 0:
                net = slim.conv2d(input, n_channels, [1, ksz], activation_fn=lrelu,
                                  normalizer_fn=norm_fn, scope='se_conv_%d' % id,
                                  padding='SAME', reuse=reuse)
            else:
                net, pad_elements = signal_to_dilated(net, n_channels=n_channels,
dilation=2 ** id)
                net = slim.conv2d(net, n_channels, [1, ksz], activation_fn=lrelu,
                                  normalizer_fn=norm_fn, scope='se_conv_%d' % id,
                                  padding='SAME', reuse=reuse)
                net = dilated_to_signal(net, n_channels=n_channels,
pad_elements=pad_elements)

        net = slim.conv2d(net, n_channels, [1, ksz], activation_fn=lrelu,
                          normalizer_fn=norm_fn, scope='se_conv_last',
                          padding='SAME', reuse=reuse)

        output = slim.conv2d(net, 1, [1, 1], activation_fn=None,
                             scope='se_fc_last', padding='SAME', reuse=reuse)

        return output
```