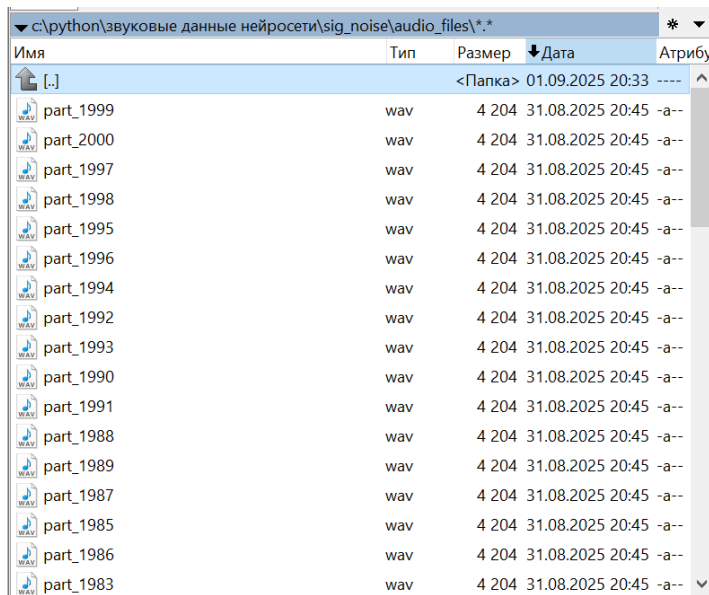


Лабораторная работа №4

Обработка звуковых данных с помощью нейросети.

Построим модель обработки с одномерным входом и выходом. В качестве входных данных будут файлы одноканальные файлы wav с добавленным случайным шумом, в качестве выходных данных будут исходные файлы.



Имя	Тип	Размер	Дата	Атрибу
[.]		<Папка>	01.09.2025 20:33	----
part_1999	wav	4 204	31.08.2025 20:45	-a--
part_2000	wav	4 204	31.08.2025 20:45	-a--
part_1997	wav	4 204	31.08.2025 20:45	-a--
part_1998	wav	4 204	31.08.2025 20:45	-a--
part_1995	wav	4 204	31.08.2025 20:45	-a--
part_1996	wav	4 204	31.08.2025 20:45	-a--
part_1994	wav	4 204	31.08.2025 20:45	-a--
part_1992	wav	4 204	31.08.2025 20:45	-a--
part_1993	wav	4 204	31.08.2025 20:45	-a--
part_1990	wav	4 204	31.08.2025 20:45	-a--
part_1991	wav	4 204	31.08.2025 20:45	-a--
part_1988	wav	4 204	31.08.2025 20:45	-a--
part_1989	wav	4 204	31.08.2025 20:45	-a--
part_1987	wav	4 204	31.08.2025 20:45	-a--
part_1985	wav	4 204	31.08.2025 20:45	-a--
part_1986	wav	4 204	31.08.2025 20:45	-a--
part_1983	wav	4 204	31.08.2025 20:45	-a--

Импорт библиотек

```
import os
import numpy as np
from scipy.io import wavfile
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, UpSampling1D, InputLayer
from tensorflow.keras.layers import Cropping1D, ZeroPadding1D
```

```
# Параметры обработки звука
SAMPLE_RATE = 16000 # Частота дискретизации
NOISE_LEVEL = 0.1 # Уровень добавляемого шума
```

```
def load_audio(filename):
    """Загрузка аудиофайла."""
    sr, audio_data = wavfile.read(filename)
    return sr, audio_data.astype(np.float32)
```

```
def add_noise(clean_signal, noise_level=NOISE_LEVEL):
    """Добавляем случайный белый шум к чистому сигналу."""
    noisy_signal = clean_signal + noise_level * np.random.normal(size=clean_signal.shape)
    return noisy_signal
```

Построение модели

```
def build_model(input_shape=(None, 1)):
    """Строим модель 1D CNN Autoencoder."""
    model = Sequential([
        InputLayer(input_shape=input_shape),
        Conv1D(filters=32, kernel_size=3, strides=1, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=64, kernel_size=3, strides=1, activation='relu', padding='same'),
        MaxPooling1D(pool_size=2),
        UpSampling1D(size=2),
        Conv1D(filters=64, kernel_size=3, strides=1, activation='relu', padding='same'),
        UpSampling1D(size=2),

        Conv1D(filters=32, kernel_size=3, strides=1, activation='relu', padding='same'),
        #Cropping1D(cropping=(1, 1)),
        ZeroPadding1D(padding=(1, 1)),
        Conv1D(filters=1, kernel_size=3, strides=1, activation='linear', padding='same')
    ])
    model.add(Cropping1D(cropping=(1, 1)))
    return model

# Настройки пути к аудиофайлам
data_dir = './audio_files'
files = os.listdir(data_dir)
noisy_signals = []
clean_signals = []

# Читаем файлы и создаем пару шумного и чистого сигналов
for file in files:
    sr, clean_audio = load_audio(os.path.join(data_dir, file))
    noisy_audio = add_noise(clean_audio)
    noisy_signals.append(noisy_audio.reshape(-1, 1)) # Преобразуем в нужный формат
    clean_signals.append(clean_audio.reshape(-1, 1))

# Проверка длин и выравнивание данных
max_len = max(len(sig) for sig in noisy_signals)
padded_noisy_signals = np.zeros((len(files), max_len, 1))
padded_clean_signals = np.zeros((len(files), max_len, 1))

for i, (noisy_sig, clean_sig) in enumerate(zip(noisy_signals, clean_signals)):
    padded_noisy_signals[i, :len(noisy_sig)] = noisy_sig
    padded_clean_signals[i, :len(clean_sig)] = clean_sig

# Создаем модель
model = build_model(input_shape=(None, 1))
model.compile(optimizer="adam", loss="mse")
```

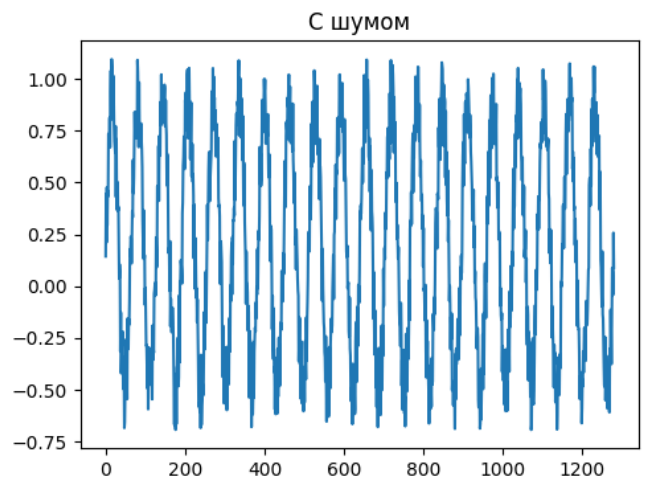
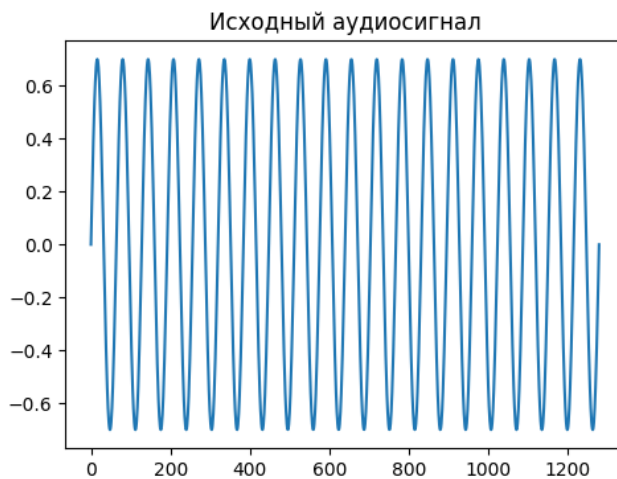
Обучение модели

```
# Обучаем модель
history = model.fit(
    x=padded_noisy_signals,
    y=padded_clean_signals,
    epochs=20,
    batch_size=4,
    validation_split=0.2
)
```

Проверочный сигнал возьмем гармонический, с добавленным шумом.

```
import numpy as np
import matplotlib.pyplot as plt
# Создаем пример аудиосигнала
N=1280
clean_signal = 0.7 * np.sin(40 * np.pi * np.linspace(0, 1, N))
noise_signal = clean_signal + 0.4 * np.random.random(N)

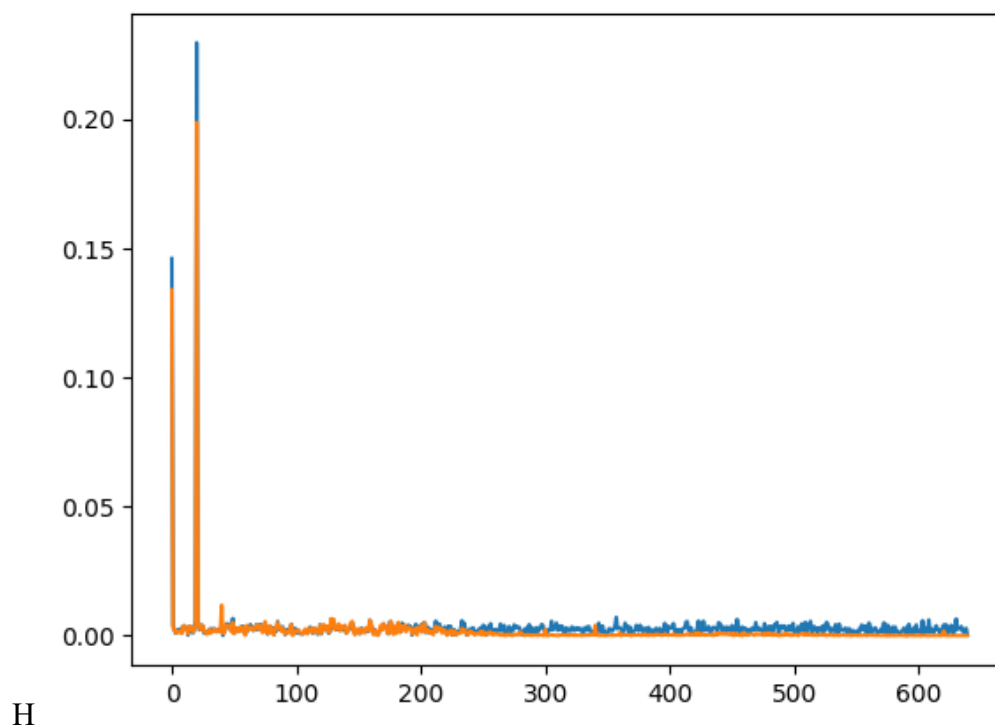
plt.figure(figsize=(12, 4))
plt.subplot(121)
plt.plot(clean_signal)
plt.title('Исходный аудиосигнал')
plt.subplot(122)
plt.plot(noise_signal) # Детализирующие коэффициенты
plt.title('С шумом')
plt.show()
```



```
reshaped_test_signal = noise_signal.reshape(1, -1, 1)
# преобразуем в нужную форму, затем можно отправить сигнал в модель для
#предсказания:
```

```
predicted_output = model.predict(reshaped_test_signal)
```

Спектры исходного сигнала (синий) и сигнала после обработки сетью (оранжевый)



Задание: применить один из цифровых фильтров из своей работы №3 к широкополосному сигналу, провести обучение и сравнить фильтрацию с помощью сети и с помощью цифрового фильтра.