МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ЗВІТ

з лабораторної роботи №8

з навчальної дисципліни «Проектування та реалізація програмних систем з

нейронними мережами»

|  |  |
| --- | --- |
| Виконав:  студент групи ІП-15  Мєшков Андрій Ігорович | Перевірив:  Шимкович В.М. |

Київ 2024

**ЛАБОРАТОРНА РОБОТА №8**

**Тема:** Нейронні мережі СNN-bi-LSTM для розпізнавання звуку

**Завдання** – Написати програму, що реалізує нейронну мережу типу СNN-bi-LSTM для розпізнавання мови в текст. Використати датасет LJ-Speech: <https://keithito.com/LJ-Speech-Dataset/>

**Хід роботи**

# Завантаження необхідних бібліотек

*import* pandas *as* pd

*import* numpy *as* np

*import* tensorflow *as* tf

*from* tensorflow *import* keras

*from* tensorflow.keras *import* layers

*import* matplotlib.pyplot *as* plt

*from* IPython *import* display

*from* jiwer *import* wer

# Завантаження набору даних LJSpeech

data\_url = "https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2"

data\_path = keras.utils.get\_file("LJSpeech-1.1", data\_url, untar=True)

wavs\_path = data\_path + "/wavs/"

metadata\_path = data\_path + "/metadata.csv"

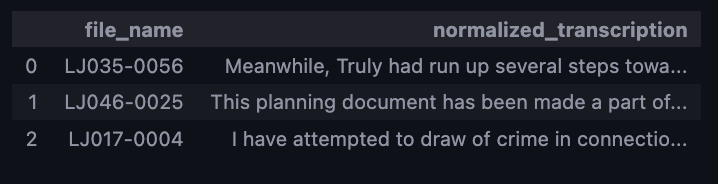
metadata\_df = pd.read\_csv(metadata\_path, sep="|", header=None, quoting=3)

metadata\_df.columns = ["file\_name", "transcription", "normalized\_transcription"]

metadata\_df = metadata\_df[["file\_name", "normalized\_transcription"]]

metadata\_df = metadata\_df.sample(frac=1).reset\_index(drop=True)

metadata\_df.head(3)



# Розбиття даних на навчальні та тренувальні дані.

split = int(len(metadata\_df) \* 0.90)

df\_train = metadata\_df[:split]

df\_val = metadata\_df[split:]

print(f"Size of the training set: {len(df\_train)}")

print(f"Size of the validation set: {len(df\_val)}")

Size of the training set: 11790

Size of the validation set: 1310

# Попередня обробка

characters = [x *for* x *in* "abcdefghijklmnopqrstuvwxyz'?! "]

char\_to\_num = keras.layers.StringLookup(vocabulary=characters, oov\_token="")

num\_to\_char = keras.layers.StringLookup(

vocabulary=char\_to\_num.get\_vocabulary(), oov\_token="", invert=True

)

print(

f"The vocabulary is: {char\_to\_num.get\_vocabulary()} "

f"(size ={char\_to\_num.vocabulary\_size()})"

)

The vocabulary is: ['', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', "'", '?', '!', ' '] (size =31)

# Функція, яка описує перетворення, яке ми застосовуємо до кожного елемента нашого набору даних.

frame\_length = 256

frame\_step = 160

fft\_length = 384

def encode\_single\_sample(wav\_file, label):

file = tf.io.read\_file(wavs\_path + wav\_file + ".wav")

audio, \_ = tf.audio.decode\_wav(file)

audio = tf.squeeze(audio, axis=-1)

audio = tf.cast(audio, tf.float32)

spectrogram = tf.signal.stft(

audio, frame\_length=frame\_length, frame\_step=frame\_step, fft\_length=fft\_length

)

spectrogram = tf.abs(spectrogram)

spectrogram = tf.math.pow(spectrogram, 0.5)

means = tf.math.reduce\_mean(spectrogram, 1, keepdims=True)

stddevs = tf.math.reduce\_std(spectrogram, 1, keepdims=True)

spectrogram = (spectrogram - means) / (stddevs + 1e-10)

label = tf.strings.lower(label)

label = tf.strings.unicode\_split(label, input\_encoding="UTF-8")

label = char\_to\_num(label)

*return* spectrogram, label

# Створення об'єктів набору даних

batch\_size = 32

train\_dataset = tf.data.Dataset.from\_tensor\_slices(

(list(df\_train["file\_name"]), list(df\_train["normalized\_transcription"]))

)

train\_dataset = (

train\_dataset.map(encode\_single\_sample, num\_parallel\_calls=tf.data.AUTOTUNE)

.padded\_batch(batch\_size)

.prefetch(buffer\_size=tf.data.AUTOTUNE)

)

validation\_dataset = tf.data.Dataset.from\_tensor\_slices(

(list(df\_val["file\_name"]), list(df\_val["normalized\_transcription"]))

)

validation\_dataset = (

validation\_dataset.map(encode\_single\_sample, num\_parallel\_calls=tf.data.AUTOTUNE)

.padded\_batch(batch\_size)

.prefetch(buffer\_size=tf.data.AUTOTUNE)

)

# Візуалізуйте дані

fig = plt.figure(figsize=(8, 5))

*for* batch *in* train\_dataset.take(1):

spectrogram = batch[0][0].numpy()

spectrogram = np.array([np.trim\_zeros(x) *for* x *in* np.transpose(spectrogram)])

label = batch[1][0]

label = tf.strings.reduce\_join(num\_to\_char(label)).numpy().decode("utf-8")

ax = plt.subplot(2, 1, 1)

ax.imshow(spectrogram, vmax=1)

ax.set\_title(label)

ax.axis("off")

file = tf.io.read\_file(wavs\_path + list(df\_train["file\_name"])[0] + ".wav")

audio, \_ = tf.audio.decode\_wav(file)

audio = audio.numpy()

ax = plt.subplot(2, 1, 2)

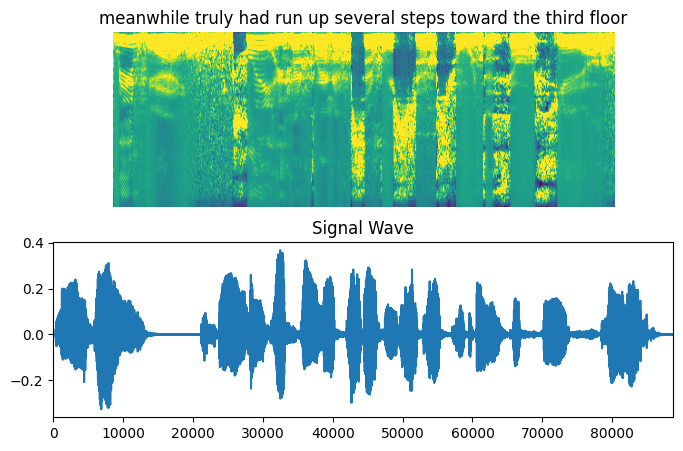
plt.plot(audio)

ax.set\_title("Signal Wave")

ax.set\_xlim(0, len(audio))

display.display(display.Audio(np.transpose(audio), rate=16000))

plt.show()



# Модель

def CTCLoss(y\_true, y\_pred):

batch\_len = tf.cast(tf.shape(y\_true)[0], dtype="int64")

input\_length = tf.cast(tf.shape(y\_pred)[1], dtype="int64")

label\_length = tf.cast(tf.shape(y\_true)[1], dtype="int64")

input\_length = input\_length \* tf.ones(shape=(batch\_len, 1), dtype="int64")

label\_length = label\_length \* tf.ones(shape=(batch\_len, 1), dtype="int64")

loss = keras.backend.ctc\_batch\_cost(y\_true, y\_pred, input\_length, label\_length)

*return* loss

def build\_model(input\_dim, output\_dim, rnn\_layers=5, rnn\_units=128):

input\_spectrogram = layers.Input((None, input\_dim), name="input")

x = layers.Reshape((-1, input\_dim, 1), name="expand\_dim")(input\_spectrogram)

x = layers.Conv2D(

filters=32,

kernel\_size=[11, 41],

strides=[2, 2],

padding="same",

use\_bias=False,

name="conv\_1",

)(x)

x = layers.BatchNormalization(name="conv\_1\_bn")(x)

x = layers.ReLU(name="conv\_1\_relu")(x)

x = layers.Conv2D(

filters=32,

kernel\_size=[11, 21],

strides=[1, 2],

padding="same",

use\_bias=False,

name="conv\_2",

)(x)

x = layers.BatchNormalization(name="conv\_2\_bn")(x)

x = layers.ReLU(name="conv\_2\_relu")(x)

x = layers.Reshape((-1, x.shape[-2] \* x.shape[-1]))(x)

*for* i *in* range(1, rnn\_layers + 1):

recurrent = layers.LSTM(

units=rnn\_units,

activation="tanh",

recurrent\_activation="sigmoid",

use\_bias=True,

return\_sequences=True,

name=f"lstm\_{i}",

)

x = layers.Bidirectional(

recurrent, name=f"bidirectional\_{i}", merge\_mode="concat"

)(x)

*if* i < rnn\_layers:

x = layers.Dropout(rate=0.5)(x)

x = layers.Dense(units=rnn\_units \* 2, name="dense\_1")(x)

x = layers.ReLU(name="dense\_1\_relu")(x)

x = layers.Dropout(rate=0.5)(x)

output = layers.Dense(units=output\_dim + 1, activation="softmax")(x)

model = keras.Model(input\_spectrogram, output, name="DeepSpeech\_2")

opt = keras.optimizers.Adam(learning\_rate=1e-4)

model.compile(optimizer=opt, loss=CTCLoss)

*return* model

model = build\_model(

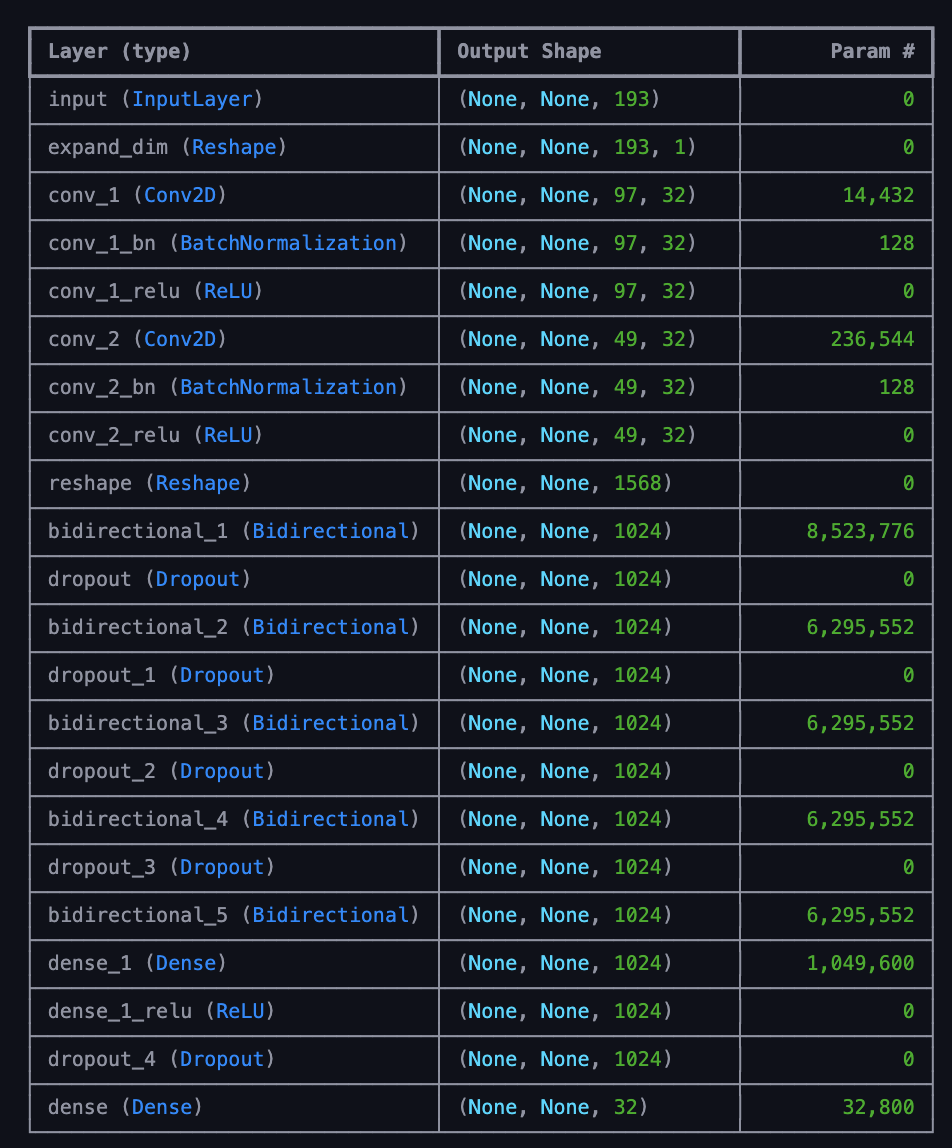
input\_dim=fft\_length // 2 + 1,

output\_dim=char\_to\_num.vocabulary\_size(),

rnn\_units=512,

)

model.summary()



def decode\_batch\_predictions(pred):

input\_len = np.ones(pred.shape[0]) \* pred.shape[1]

results = keras.backend.ctc\_decode(pred, input\_length=input\_len, greedy=True)[0][0]

output\_text = []

*for* result *in* results:

result = tf.strings.reduce\_join(num\_to\_char(result)).numpy().decode("utf-8")

output\_text.append(result)

*return* output\_text

epochs = 10

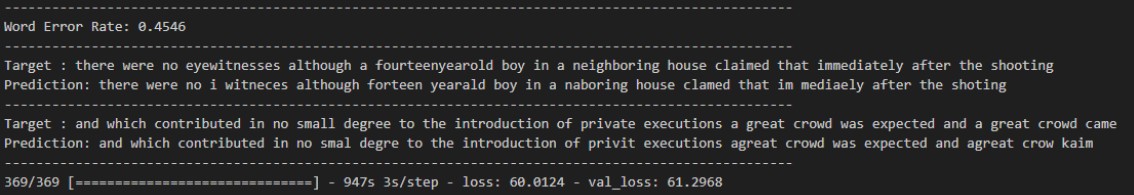
history = model.fit(

train\_dataset,

validation\_data=validation\_dataset,

epochs=epochs,

)



**Висновок:**

Під час виконання лабораторної роботи було реалізовано нейронну мережу типу СNN-bi-LSTM для розпізнавання мови в текст.