НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіт до комп’ютерного практикуму 8 з дисципліни: “**Програмні засоби проектування та реалізації нейромережевих систем**”

**Виконав**

**ІП-01 Черпак А.В.**

**Перевірив:**

**Шимкович В.М.**

Київ – 2023

**Комп’ютерний практикум 4**

**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 librispeech: <https://www.tensorflow.org/datasets/catalog/librispeech>

**Виконання:**

Створимо методи для побудови моделі та створення функції втрат CTC:

from keras.layers import ReLU, Dense, LSTM, Dropout, BatchNormalization, Conv2D, Reshape, Input, Bidirectional  
import tensorflow as tf  
  
  
def buildModel(input\_dim, output\_dim, numOfRNN, numOfRNNUnits):  
 input = Input((None, input\_dim))  
 x = Reshape((-1, input\_dim, 1))(input)  
 x = Conv2D(filters=32, kernel\_size=[11, 41], strides=[2, 2], padding="same", use\_bias=False)(x)  
 x = BatchNormalization()(x)  
 x = ReLU()(x)  
 x = Conv2D(filters=32, kernel\_size=[11, 21], strides=[1, 2], padding="same", use\_bias=False)(x)  
 x = BatchNormalization()(x)  
 x = ReLU()(x)  
 x = Reshape((-1, x.shape[-2] \* x.shape[-1]))(x)  
 for i in range(numOfRNN):  
 recurrent = LSTM(units=numOfRNNUnits, activation="tanh", recurrent\_activation="sigmoid", use\_bias=True,  
 return\_sequences=True)  
 x = Bidirectional(recurrent, merge\_mode="concat")(x)  
 if i < numOfRNN - 1:  
 x = Dropout(rate=0.5)(x)  
  
 x = Dense(numOfRNNUnits \* 2)(x)  
 x = ReLU()(x)  
 x = Dropout(rate=0.5)(x)  
 output = Dense(output\_dim + 1, activation="softmax")(x)  
 model = tf.keras.Model(input, output)  
 model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=1e-4),  
 loss=CTCLoss)  
 return model  
  
  
def CTCLoss(y\_true, y\_pred):  
 batchLen = tf.cast(tf.shape(y\_true)[0], "int64")  
 inputLen = tf.cast(tf.shape(y\_pred)[1], "int64") \* tf.ones(shape=(batchLen, 1), dtype="int64")  
 labelLen = tf.cast(tf.shape(y\_true)[1], "int64") \* tf.ones(shape=(batchLen, 1), dtype="int64")  
 loss = tf.keras.backend.ctc\_batch\_cost(y\_true, y\_pred, inputLen, labelLen)  
 return loss

Створимо метод для декодування передбачень, а також колбек, який наприкінці кожної епохи візуалізуватиме результат тестування:

import tensorflow as tf  
import numpy as np  
from jiwer import wer  
  
characters = [x for x in "abcdefghijklmnopqrstuvwxyz'?! "]  
charToNum = tf.keras.layers.StringLookup(vocabulary=characters, oov\_token='')  
numToChar = tf.keras.layers.StringLookup(vocabulary=charToNum.get\_vocabulary(), oov\_token='', invert=True)  
  
  
def decodePredictions(pred):  
 inputLen = np.ones(pred.shape[0]) \* pred.shape[1]  
 results = tf.keras.backend.ctc\_decode(pred, input\_length=inputLen, greedy=True)[0][0]  
 outputText = [tf.strings.reduce\_join(numToChar(result)).numpy().decode('utf-8') for result in results]  
 return outputText  
  
  
class CallbackEval(tf.keras.callbacks.Callback):  
 def \_\_init\_\_(self, dataset, model):  
 super().\_\_init\_\_()  
 self.dataset = dataset  
 self.model = model  
  
 def on\_epoch\_end(self, epoch: int, logs=None):  
 predictions = []  
 targets = []  
 for spectrograms, labels in self.dataset:  
 batch\_predictions = self.model.predict(spectrograms, verbose=0)  
 batch\_predictions = decodePredictions(batch\_predictions)  
 predictions.extend(batch\_predictions)  
 for label in labels:  
 label = (tf.strings.reduce\_join(numToChar(label)).numpy().decode('utf-8'))  
 targets.append(label)  
 wer\_score = wer(targets, predictions)  
 print('-' \* 100)  
 print(f'Word Error Rate: {wer\_score:.4f}')  
 print('-' \* 100)  
 for i in np.random.randint(0, len(predictions), 2):  
 print(f'Target : {targets[i]}')  
 print(f'Prediction: {predictions[i]}')  
 print('-' \* 100)

У головному файлі також створимо функцію для обробки кожної точки даних, тобто переведення аудіо у спектрограму, а текст – у набір міток:

import tensorflow\_datasets as tfds  
import tensorflow as tf  
import numpy as np  
import matplotlib.pyplot as plt  
  
from Lab8.CallbackEval import CallbackEval  
from nn\_model import buildModel  
  
  
def processSample(label, audio):  
 audio = tf.cast(audio, tf.float32)  
 spectr = tf.signal.stft(audio, frame\_length=256, frame\_step=160, fft\_length=fft\_length)  
 spectr = tf.math.pow(tf.math.abs(spectr), 0.5)  
 mean = tf.math.reduce\_mean(spectr, 1, keepdims=True)  
 std = tf.math.reduce\_std(spectr, 1, keepdims=True)  
 spectr = (spectr - mean) / (std + 1e-10)  
 label = charToNum(tf.strings.unicode\_split(tf.strings.lower(label), input\_encoding='UTF-8'))  
 return spectr, label

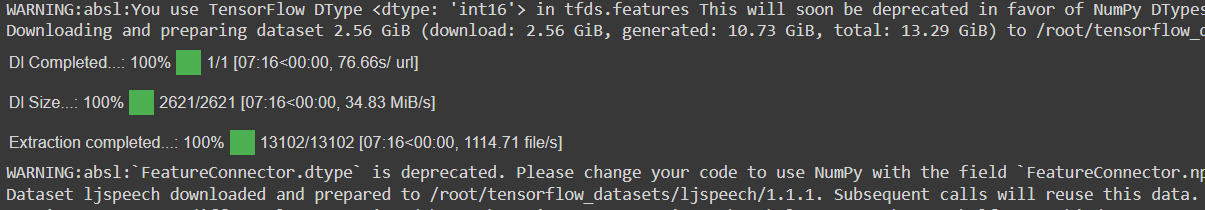
Тепер у розділі \_\_main\_\_ завантажимо необхідний датасет, одразу розділимо його на тренувальну та тесту вальну вибірки. Також визначимо набір символів, які розпізнаватимемо. Виведемо текст одного з семплів, а також візуалізуємо його спектрограму. Потім створимо модель, виведемо дані про неї та проведемо 5 епох навчання.

if \_\_name\_\_ == '\_\_main\_\_':  
 trainData, testData = tfds.load("ljspeech", split=["train[:90%]", "train[90%:]"], as\_supervised=True)  
 characters = [x for x in "abcdefghijklmnopqrstuvwxyz'?! "]  
 charToNum = tf.keras.layers.StringLookup(vocabulary=characters, oov\_token='')  
 numToChar = tf.keras.layers.StringLookup(vocabulary=charToNum.get\_vocabulary(), oov\_token='', invert=True)  
 SAMPLE\_RATE = 22050  
 fft\_length = 384  
  
 trainPreprocessed = (trainData.map(processSample, num\_parallel\_calls=tf.data.AUTOTUNE).padded\_batch(32).prefetch(buffer\_size=tf.data.AUTOTUNE))  
 testPreprocessed = (testData.map(processSample, num\_parallel\_calls=tf.data.AUTOTUNE).padded\_batch(32).prefetch(buffer\_size=tf.data.AUTOTUNE))  
 for spectrograms, labels in trainPreprocessed.take(1):  
 spectrogram = spectrograms[1].numpy()  
 spectrogram = np.array([np.trim\_zeros(x) for x in np.transpose(spectrogram)])  
 label = labels[1]  
 label = tf.strings.reduce\_join(numToChar(label)).numpy().decode('utf-8')  
 plt.imshow(spectrogram, vmax=1)  
 print(label)  
  
 model = buildModel(fft\_length // 2 + 1, charToNum.vocabulary\_size(), 5, 512)  
 model.summary()  
  
 callback = CallbackEval(testPreprocessed, model)  
 history = model.fit(trainPreprocessed, validation\_data=testPreprocessed, epochs=5, callbacks=[callback])  
 callback.on\_epoch\_end(5)  
  
 model.save('/model.h5')

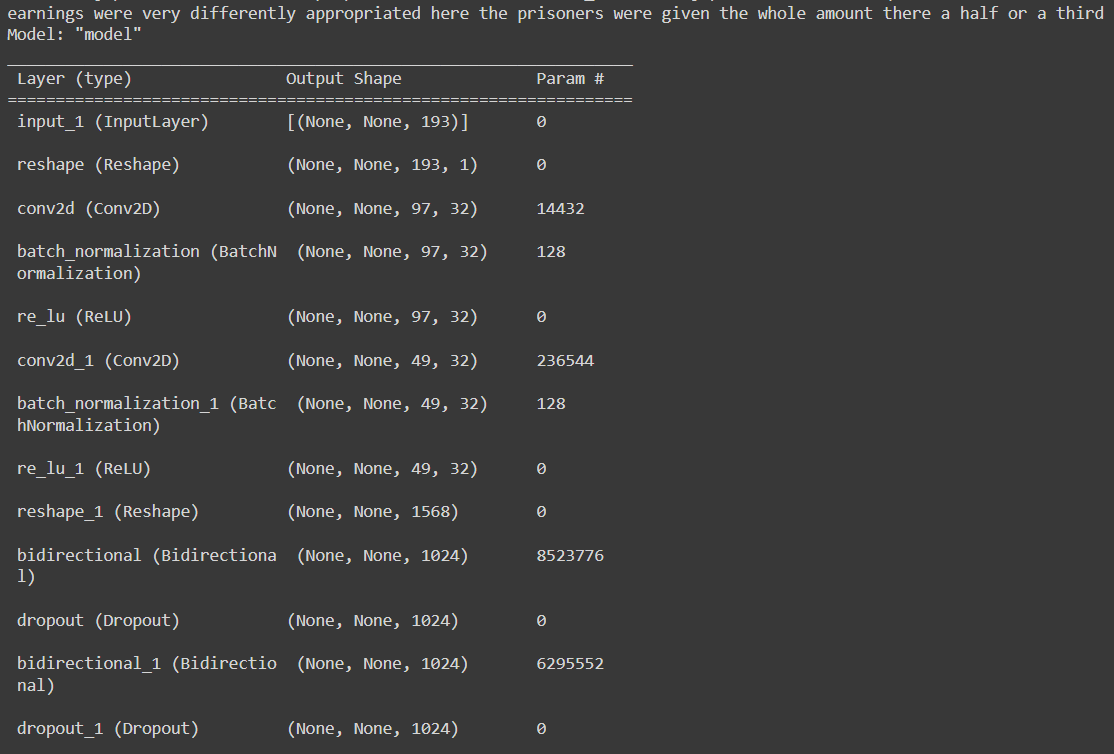
Для тренування нейромережі було використано Google Colab, оскільки на моєму пристрої тренування проходило нереально повільно.

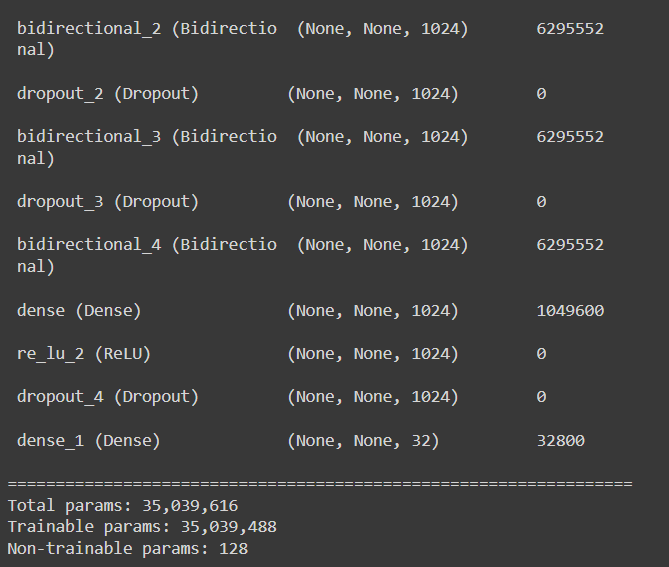
Тепер наведемо результати виконання даного коду.

Результати завантаження датасету:

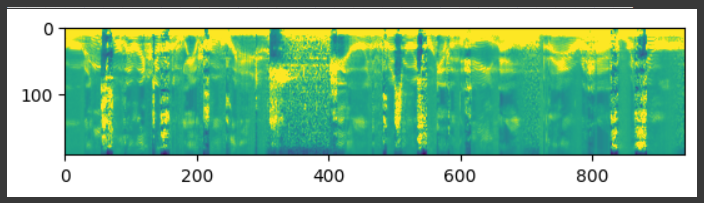


Текст одного з семплів та дані про модель:

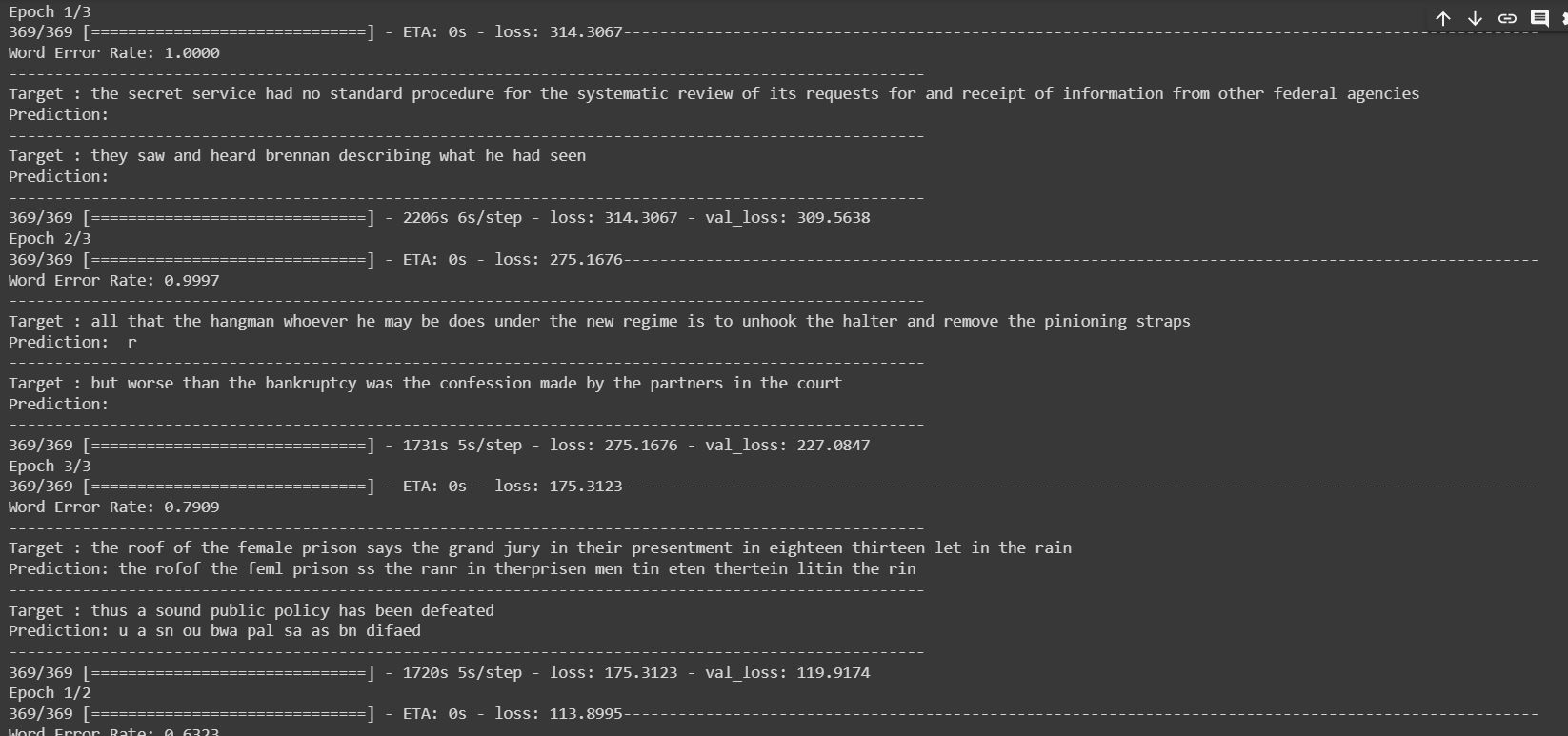




Спектрограма одного з семплів:



Результати тренування моделі:



Про всяк випадок після трьох епох тренування я зберіг модень, а потім провів ще 2 епохи навчання.



Як бачимо, вже після 5 епох точність досягла майже 50%, що насправді непогано для такої складної задачі. І, як бачимо з першого семплу з останнього тестування, насправді помилок геть небагато, і більшість з них незначні. А з останнім семплом пощастило менше.

**Висновок:**

Під час виконання комп’ютерного практикуму ми реалізували гібридну двонаправлену CNN-biLSTM нейромережу для розпізнавання тексту з уривків звукозаписів із датасету ljspeech. Після 5 епох тренувань ми отримали точність 47,21%, при чому, деякі уривки були розпізнані з похибкою буквально у три літери, що насправді дуже навіть непогано. Щоправда, її тренування на моєму ноутбуці проходило вкрай повільно, тому довелося використовувати google colab з підтримкою GPU. У такому випадку тренування тривало близько 4 годин.