Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем і технологій

Лабораторна робота №8

Проектування та реалізація програмних систем з нейронними мережами

Нейронні мережі CNN-bi-LSTM для розпізнавання звуку

Завдання: Написати програму, що реалізує нейронну мережу типу CNN-bi-L

Програмний код:

```
# підключення необхідних бібліотек для навчання
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
from keras.utils.vis_utils import plot_model
from IPython import display

seed = 12
tf.random.set_seed(seed)
np.random.seed(seed)

# розташування датасету
DATASET_PATH = 'data/mini_speech_commands'

data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
  tf.keras.utils.get_file(
      'mini_speech_commands.zip',

origin="http://storage.googleapis.com/download.tensorflow.org/data/mini_speech_commands.zip",
      extract=True,
      cache_dir='.', cache_subdir='data')

"""The dataset's audio clips are stored in eight folders corresponding to each speech command: `no`, `yes`, `down`, `go`, `left`, `up`, `right`, and `stop`:"""

# завантаження переліку команд
commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[commands != 'README.md']
commands

# завантаження датасету і вивід його об'єму
filenames = tf.io.gfile.glob(str(data_dir) + '/*/*')
filenames = tf.random.shuffle(filenames)
num_samples = len(filenames)
print('Number of total examples:', num_samples)
print('Number of examples per label:',
      len(tf.io.gfile.listdir(str(data_dir/commands[0]))))
```

Т

е

к

```
print('Example file:', filenames[0])

"""## Read the audio files and their labels"""

# зчитування аудіофайлів та перевірка їх розмірності
test_file =
tf.io.read_file(DATASET_PATH+'/down/0a9f9af7_nohash_0.wav')
test_audio, _ = tf.audio.decode_wav(contents=test_file)
test_audio.shape

"""Now, let's define a function that preprocesses the dataset's
raw WAV audio files into audio tensors:"""

# перетворення WAV аудіофайлів в аудіотензори
def decode_audio(audio_binary):
  audio, _ = tf.audio.decode_wav(contents=audio_binary)
  return tf.squeeze(audio, axis=-1)

"""Define a function that creates labels using the parent
directories for each file:"""

# створення міток за допомогою шляхів до файлів
def get_label(file_path):
  parts = tf.strings.split(
      input=file_path,
      sep=os.path.sep)
  return parts[-2]

# отримання міток та декодування аудіо
def get_waveform_and_label(file_path):
  label = get_label(file_path)
  audio_binary = tf.io.read_file(file_path)
  waveform = decode_audio(audio_binary)
  return waveform, label

# розподіл датасету на тренувальний, валідаційний та тестовий
train_files = filenames[:6400]
val_files = filenames[6400: 6400 + 800]
test_files = filenames[-800:]
AUTOTUNE = tf.data.AUTOTUNE

# формування датасету
files_ds = tf.data.Dataset.from_tensor_slices(train_files)

waveform_ds = files_ds.map(
    map_func=get_waveform_and_label,
    num_parallel_calls=AUTOTUNE)

# виведення аудіодоріжок, що відповідають аудіо з датасету
rows = 3
cols = 3
n = rows * cols
fig, axes = plt.subplots(rows, cols, figsize=(10, 12))
```

```
for i, (audio, label) in enumerate(waveform_ds.take(n)):
  r = i // cols
  c = i % cols
  ax = axes[r][c]
  ax.plot(audio.numpy())
  ax.set_yticks(np.arange(-1.2, 1.2, 0.2))
  label = label.numpy().decode('utf-8')
  ax.set_title(label)

plt.show()

# отримання спектрограми
def get_spectrogram(waveform):
  input_len = 16000  # довжина доріжки
  waveform = waveform[:input_len]
  zero_padding = tf.zeros(
      [16000] - tf.shape(waveform),
      dtype=tf.float32)
  waveform = tf.cast(waveform, dtype=tf.float32) # перетворення
типу
  equal_length = tf.concat([waveform, zero_padding], 0) #
конкатенація тензорів
  spectrogram = tf.signal.stft(
      equal_length, frame_length=255, frame_step=128) #
перетворення Фур'є для сигналів
  spectrogram = tf.abs(spectrogram) # модуль тензорів
  spectrogram = spectrogram[..., tf.newaxis] # отримання
потрібної спектрограми
  return spectrogram

# отримання спектрограми і мітки
def get_spectrogram_and_label_id(audio, label):
  spectrogram = get_spectrogram(audio)
  label_id = tf.math.argmax(label == commands)
  return spectrogram, label_id

# створення map для спектрограм і міток
spectrogram_ds = waveform_ds.map(
  map_func=get_spectrogram_and_label_id,
  num_parallel_calls=AUTOTUNE)

"""## Build and train the model

Repeat the training set preprocessing on the validation and test
sets:
"""

# підготовка датасету
def preprocess_dataset(files):
  files_ds = tf.data.Dataset.from_tensor_slices(files) # файли
датасету
  output_ds = files_ds.map(
```

```
      map_func=get_waveform_and_label,
      num_parallel_calls=AUTOTUNE)
  output_ds = output_ds.map(
      map_func=get_spectrogram_and_label_id,
      num_parallel_calls=AUTOTUNE)
  return output_ds # вихідний датасет

# підготовка навчальних, валідаційних та тестових даних
train_ds = spectrogram_ds
val_ds = preprocess_dataset(val_files)
test_ds = preprocess_dataset(test_files)

"""Batch the training and validation sets for model training:"""

batch_size = 64 # розмір вибірки
# розбиття на вибірки
train_ds = train_ds.batch(batch_size)
val_ds = val_ds.batch(batch_size)

"""Add `Dataset.cache` and `Dataset.prefetch` operations to
reduce read latency while training the model:"""

# додавання кешу і попередньої вибірки для зменшення затримки
під час навчання
train_ds = train_ds.cache().prefetch(AUTOTUNE)
val_ds = val_ds.cache().prefetch(AUTOTUNE)

# побудова моделі
def cnn_lstm(input_dim=26, filters=1024, rnn_size=512,
output_dim=29, convolutional_layers=3, lstm_layers=5): # 26
вхідних, 1024 фільтри, 512 - розмір рекурентної мережі, 29 -
вихідних, 3 згорткові шари, 5 запам'ятовуючих шарів
    input_layer = layers.Input(name='input', shape=(None,
input_dim)) # вхідний шар

    # нормалізація і "оббивка"
    layer = layers.BatchNormalization(axis=-1)(input_layer)
    layer = layers.ZeroPadding1D(padding=(0, 512))(layer)

    # Додавання згорткових шарів
    for l in range(convolutional_layers):
        layer = layers.Conv1D(filters=filters,
                              name='convolution_{}'.format(l + 1),
                              kernel_size=11,
                              padding='valid',
                              activation='relu',
                              strides=2)(layer)

    layer = layers.BatchNormalization(axis=-1)(layer) #
нормалізація

    # додавання запам'ятовуючих шарів
    for l in range(lstm_layers):
```

```
        layer = layers.Bidirectional(layers.LSTM(rnn_size,
return_sequences=True))(layer)
        layer =
layers.TimeDistributed(layers.Dense(output_dim))(layer) #
часорозподільчий шар

    layer = layers.BatchNormalization(axis=-1)(layer) #
нормалізація

    # повнозв'язні шари
    layer = layers.TimeDistributed(layers.Dense(filters,
activation='relu'))(layer) # часорозподільчий шар
    prediction_layer =
layers.TimeDistributed(layers.Dense(output_dim, name="y_pred",
activation="softmax"))(layer) # часорозподільчий шар

    # вихідний шар
    model = models.Model(inputs=input_layer,
outputs=prediction_layer)
    model.output_length = lambda layer: layer

    return model

model = cnn_lstm() # генерація моделі
model.summary() # опис моделі

plot_model(model, "model.png", show_shapes=True) # графічне
подання структури мережі

# компіляція моделі
model.compile(
    optimizer=tf.keras.optimizers.Adam(), # оптимізатор Adam

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=T
rue), # функція втрат
    metrics=['accuracy'], # метрика точність
)

"""Train the model over 10 epochs for demonstration purposes:"""

EPOCHS = 10 # 10 епох навчання
# задання параметрів навчання і навчання моделі
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=tf.keras.callbacks.EarlyStopping(verbose=1,
patience=2),
)

# вивід графіка функції втрат і функції валідаційних втрат під
час навчання
metrics = history.history
```

```
plt.plot(history.epoch, metrics['loss'], metrics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.show()

"""## Evaluate the model performance

Run the model on the test set and check the model's performance:
"""

test_audio = [] # тестові аудіозаписи
test_labels = [] # тестові мітки

# підготовка тестових даних для тестування моделі
for audio, label in test_ds:
  test_audio.append(audio.numpy())
  test_labels.append(label.numpy())

test_audio = np.array(test_audio)
test_labels = np.array(test_labels)

y_pred = np.argmax(model.predict(test_audio), axis=1) #
передбачення моделі
y_true = test_labels # правильні відповіді

test_acc = sum(y_pred == y_true) / len(y_true) # точність моделі
на тестових даних
print(f'Test set accuracy: {test_acc:.0%}')

"""## Run inference on an audio file

Finally, verify the model's prediction output using an input
audio file of someone saying "no". How well does your model
perform?
"""

sample_file = data_dir/'no/01bb6a2a_nohash_0.wav' # файл з
прикладом

sample_ds = preprocess_dataset([str(sample_file)]) # підготовка
даних

for spectrogram, label in sample_ds.batch(1):
  prediction = model(spectrogram) # передбачення
  plt.bar(commands, tf.nn.softmax(prediction[0])) # стовпчаста
діаграма
  plt.title(f'Predictions for "{commands[label[0]]}"') # підписи
до діаграми
  plt.show() # вивід графіка
```

Результати виконання:

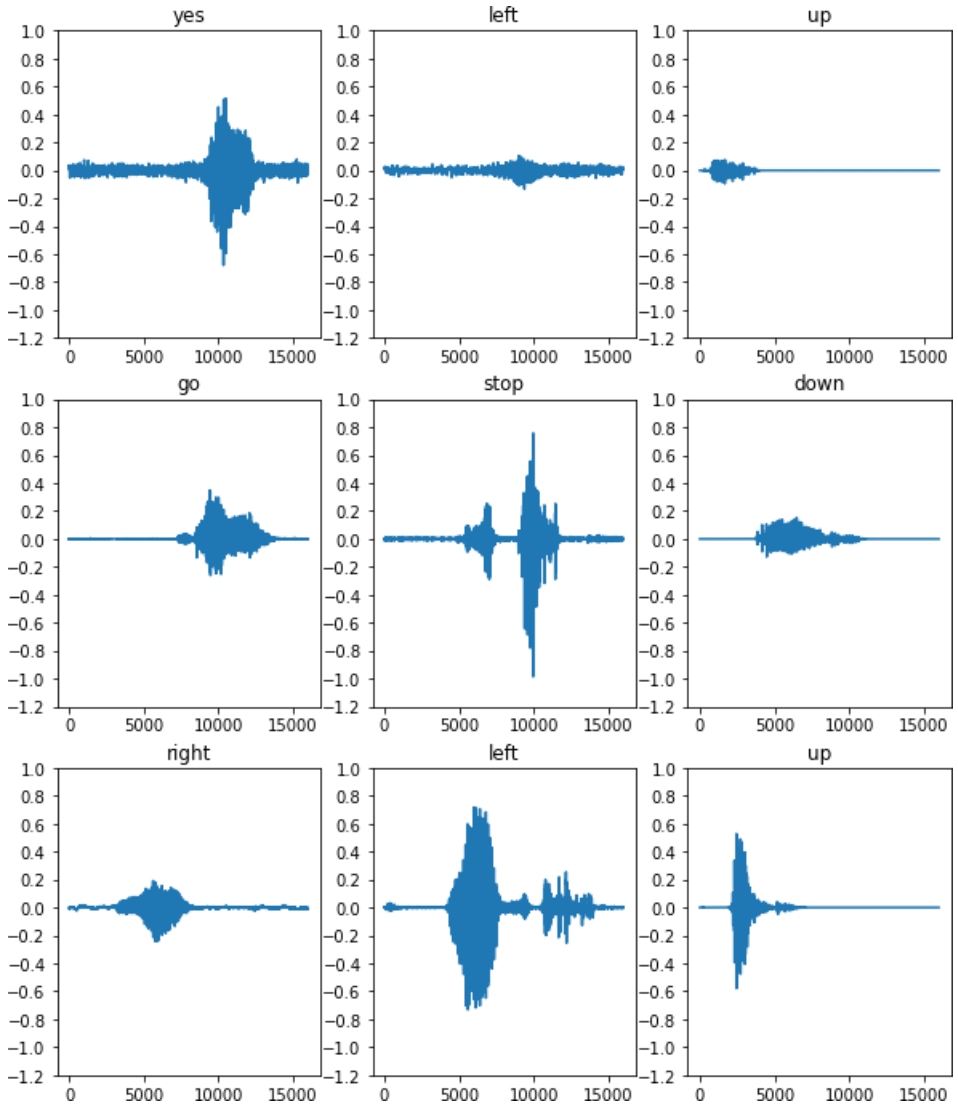

Рис. 1

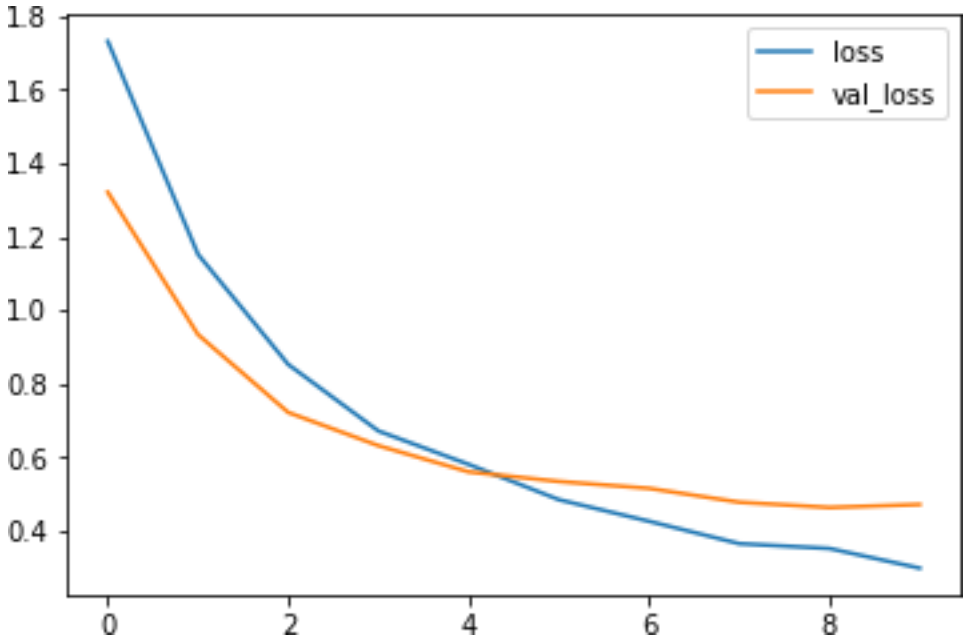

Рис. 2

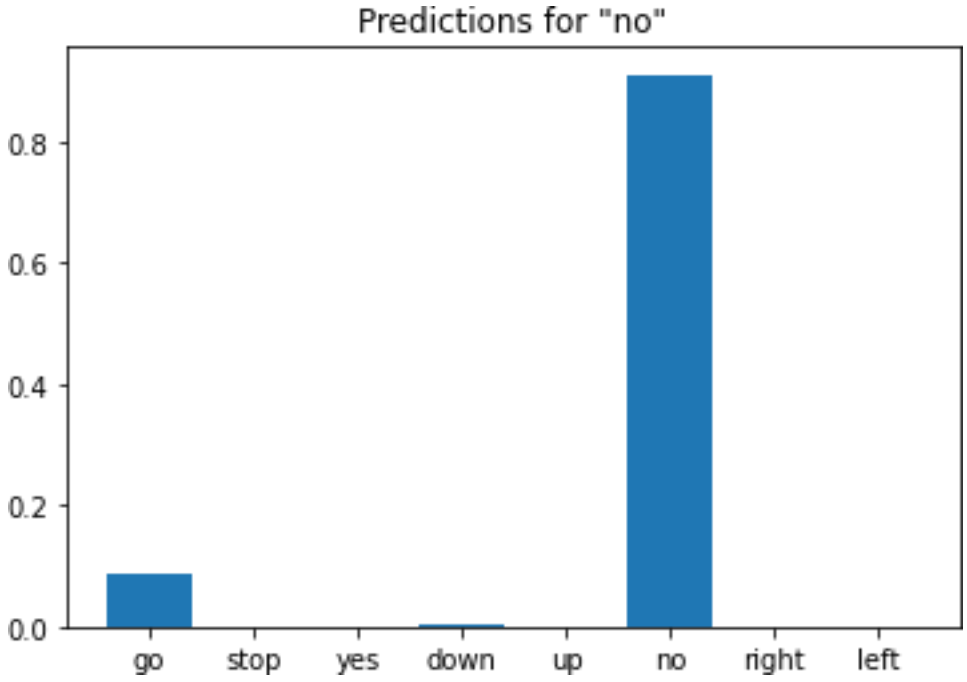

Рис. 3

Висновки

Під час виконання роботи я познайомився із побудовою рекурентної мережі та побудував мережу CNN-bi-LSTM. Також навчився розпізнавати мову в текст. Під час виконання особливих проблем не виникло. Поставлена мета успішно виконана.