ДОДАТОК А

Код програми

Нейронна мережа:

```
import os
import numpy as np
import scipy.misc
from keras import backend as K
from keras import initializers
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import np_utils
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)



from sklearn.model_selection import train_test_split

num_classes = 75
data_augmentation = True
epochs = 2
batch_size=16

img_rows, img_cols = 64, 64

earry_stopping=EarlyStopping(monitor='val_loss',patience=10, verbose=1)

save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'hirahanaMod1.h5'
```

```
ary = np.load("./dataset/etlgtobfutoji64.npz")['arr_0'].reshape([-1, 64,
64]).astype(np.float32) / 15
X_train = np.zeros([num_classes * 160, img_rows, img_cols],
dtype=np.float32)
for i in range(num_classes * 160):
    X_train[i] = scipy.misc.imresize(ary[i], (img_rows, img_cols),
mode='F')

Y_train = np.repeat(np.arange(num_classes), 160)

X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train,
test_size=0.2)

if K.image_data_format() == 'channels_first':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

Y_train = np_utils.to_categorical(Y_train, num_classes)
Y_test = np_utils.to_categorical(Y_test, num_classes)

datagen = ImageDataGenerator(rotation_range=15, zoom_range=0.20)
datagen.fit(X_train)

model = Sequential()


def my_init(mean, stddev, seed):


    return initializers.TruncatedNormal(mean=mean, stddev=stddev,
seed=seed)
my_init = my_init(mean=0.0, stddev=0.05, seed=None)
```

```
def m6_1():

    model.add(Conv2D(32, kernel_size=(3,
3),kernel_initializer=my_init,input_shape=X_train.shape[1:]))
    model.add(Activation('relu'))
    model.add(Conv2D(32, (3, 3),kernel_initializer=my_init))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))


    model.add(Conv2D(64, (3, 3),kernel_initializer=my_init))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3, 3),kernel_initializer=my_init))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))


    model.add(Flatten())
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))


def classic_neural():
    model.add(Flatten(input_shape=input_shape))
    model.add(Dense(256))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    model.add(Dense(num_classes))
    model.add(Activation('softmax'))
```

```
m6_1()

model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adadelta',
metrics=['accuracy'])




if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(X_train, Y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(X_test, Y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')

    datagen = ImageDataGenerator(
        featurewise_center=False,

        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False
        rotation_range=15,
 (degrees, 0 to 180)
        width_shift_range=0.05,  # randomly shift images horizontally
(fraction of total width)
        height_shift_range=0.05,  # randomly shift images vertically
(fraction of total height)
        horizontal_flip=False,  # randomly flip images. We should set
False!
        vertical_flip=False, # randomly flip images
        channel_shift_range=100,# change color
        samplewise_center=False)  # Normalization

applied).
    datagen.fit(X_train)
```

```
    model.fit_generator(datagen.flow(X_train, Y_train,
batch_size=batch_size), steps_per_epoch=X_train.shape[0],
                        epochs=epochs, validation_data=(X_test,
Y_test),callbacks=[earry_stopping],workers=4,initial_epoch=0)
```

```
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

scores = model.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Перевірка роботи моделі:

```
import os
import operator
from PIL import Image, ImageOps
import numpy as np
import scipy.misc
from scipy import ndimage
from keras import backend as K

from keras import initializers

from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten

from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
```

```
from keras.utils import np_utils
from keras.callbacks import EarlyStopping

from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.metrics import confusion_matrix



from keras.models import load_model
from scipy import misc

from keras.preprocessing import image

import matplotlib.pyplot as plt

import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)




kana=['あ','い','う','え','お','か','が','き','ゃ','ぎ','く','ぐ','け','げ
','こ','ご','さ','ざ','し','ゅ','じ','す','ず','せ','ぜ','そ','ぞ','た','だ
','ち','ょ','ぢ','つ','づ','て','で','と','ど','な','に','っ','ぬ','ね','の
','は','ば','ぱ','ひ','び','ぴ','ふ','ぶ','ぷ','へ','べ','ぺ','ほ','ぼ','ぽ
','ま','み','む','め','も','や','ゆ','よ','ら','り','る','れ','ろ','わ','を
','ん']
kana_num = len(kana)
print(kana_num)
labels = np.empty([0, kana_num], np.int)

num_classes = 75
data_augmentation = True
epochs = 12
batch_size=16
img_rows, img_cols = 64, 64
```

```
#img_rows, img_cols = 127, 128



l_model = load_model('./saved_models/hirahanaMod1.h5')


ary = np.load("./dataset/etlgtobfutoji64.npz")['arr_0'].reshape([-1, 64,
64]).astype(np.float32) / 15
X_train = np.zeros([num_classes * 160, img_rows, img_cols],
dtype=np.float32)
for i in range(num_classes * 160):

    X_train[i] = scipy.misc.imresize(ary[i], (img_rows, img_cols),
mode='F')
plt.imshow(ary[160*1+150])
Y_train = np.repeat(np.arange(num_classes), 160)

X_train, X_test, Y_train, Y_test = train_test_split(X_train, Y_train,
test_size=0.2)


if K.image_data_format() == 'channels_first':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

Y_train = np_utils.to_categorical(Y_train, num_classes)
Y_test = np_utils.to_categorical(Y_test, num_classes)


print([X_test[0]])
print([X_test.shape[0]])#2400
print([Y_test][0])
```

```
print([Y_test.shape[0]])#2400



images = np.empty([0, 64, 64], np.float32)
img_ori = Image.open('./sample_images/a.png')


resize_img = img_ori.resize((64, 64),Image.LANCZOS)
#img_gray = ImageOps.grayscale(resize_img)
img_gray=resize_img.convert("L")
img_b=img_gray.point(lambda x: 0 if x < 192 else x)
img_b=img_gray.point(lambda x: 255 if x >= 192 else x)
img_b.show()
img_ary = np.asarray(img_b)

img_ary = 255 - img_ary
img_ary[img_ary<110]=0
img_ary[img_ary>=110]=255

#This here, we apply to it a process of erosion for making letter thin
,because of


img_ary=ndimage.binary_erosion(img_ary)
images = np.append(images, [img_ary], axis=0)
plt.imshow(img_ary)
plt.show()

images = images.reshape(1, 64, 64, 1)

ret = l_model.predict(images, batch_size=16, verbose=0)
#print(ret)

index=np.argmax(ret, axis = None, out = None)


for i in range(len(ret)):
```

```
    #print(np.argsort(ret)[:, ::-1][i],np.sort(ret)[:, ::-1][i])
    ind=np.argsort(ret)[:, ::-1][i]
    sco=np.sort(ret)[:, ::-1][i]


for i in range(len(ind)):
    print(i+1, kana[ind[i]],round(sco[i]*100,2),'%')
    if i == 3:
      break

print('Result: '+kana[index])
```

Функція для тестування нейронної мережі:

```
package org.tensorflow.lite.codelabs.digitclassifier

import android.content.Context
import android.content.res.AssetManager
import android.graphics.Bitmap
import android.util.Log
import com.google.android.gms.tasks.Task
import com.google.android.gms.tasks.TaskCompletionSource
import java.io.FileInputStream
import java.io.IOException
import java.nio.ByteBuffer
import java.nio.ByteOrder
import java.nio.channels.FileChannel
import java.util.concurrent.ExecutorService
import java.util.concurrent.Executors
import org.tensorflow.lite.Interpreter

class DigitClassifier(private val context: Context) {
  private var interpreter: Interpreter? = null
  var isInitialized = false
    private set

  private val executorService: ExecutorService =
Executors.newCachedThreadPool()
```

```
  private var inputImageWidth: Int = 0
  private var inputImageHeight: Int = 0
  private var modelInputSize: Int = 0

  fun initialize(): Task<Void> {
    val task = TaskCompletionSource<Void>()
    executorService.execute {
      try {
        initializeInterpreter()
        task.setResult(null)
      } catch (e: IOException) {
        task.setException(e)
      }
    }
    return task.task
  }

  @Throws(IOException::class)
  private fun initializeInterpreter() {

    val assetManager = context.assets
    val model = loadModelFile(assetManager, "hiragana.tflite")
    val interpreter = Interpreter(model)

    val inputShape = interpreter.getInputTensor(0).shape()
    inputImageWidth = inputShape[1]
    inputImageHeight = inputShape[2]
    modelInputSize = FLOAT_TYPE_SIZE * inputImageWidth *
      inputImageHeight * PIXEL_SIZE
    this.interpreter = interpreter

    isInitialized = true
    Log.d(TAG, "Initialized TFLite interpreter.")
  }

  @Throws(IOException::class)
  private fun loadModelFile(assetManager: AssetManager, filename:
String): ByteBuffer {
```

```
    val fileDescriptor = assetManager.openFd(filename)
    val inputStream = FileInputStream(fileDescriptor.fileDescriptor)
    val fileChannel = inputStream.channel
    val startOffset = fileDescriptor.startOffset
    val declaredLength = fileDescriptor.declaredLength
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength)
  }

  private fun classify(bitmap: Bitmap): String {
    check(isInitialized) { "TF Lite Interpreter is not initialized yet."
}
    val resizedImage = Bitmap.createScaledBitmap(
      bitmap,
      inputImageWidth,
      inputImageHeight,
      true
    )
    val byteBuffer = convertBitmapToByteBuffer(resizedImage)

    val output = Array(1) { FloatArray(OUTPUT_CLASSES_COUNT) }

    interpreter?.run(byteBuffer, output)

    val result = output[0]
    val maxIndex = result.indices.maxByOrNull { result[it] } ?: -1
    val resultString =
      "Prediction Result: %s, %d\nConfidence: %2f"
        .format(label[maxIndex], maxIndex + 1, result[maxIndex])

    return resultString
  }

  fun classifyAsync(bitmap: Bitmap): Task<String> {
    val task = TaskCompletionSource<String>()
    executorService.execute {
      val result = classify(bitmap)
      task.setResult(result)
    }
```

```
    return task.task
  }

  fun close() {
    executorService.execute {
      interpreter?.close()
      Log.d(TAG, "Closed TFLite interpreter.")
    }
  }

  private fun convertBitmapToByteBuffer(bitmap: Bitmap): ByteBuffer {
    val byteBuffer = ByteBuffer.allocateDirect(modelInputSize)
    byteBuffer.order(ByteOrder.nativeOrder())

    val pixels = IntArray(inputImageWidth * inputImageHeight)
    bitmap.getPixels(pixels, 0, bitmap.width, 0, 0, bitmap.width,
bitmap.height)

    for (pixelValue in pixels) {
      val r = (pixelValue shr 16 and 0xFF)
      val g = (pixelValue shr 8 and 0xFF)
      val b = (pixelValue and 0xFF)

      val normalizedPixelValue = (r + g + b) / 3.0f / 255.0f
      byteBuffer.putFloat(normalizedPixelValue)
    }

    return byteBuffer
  }

  companion object {
    private const val TAG = "DigitClassifier"

    private const val FLOAT_TYPE_SIZE = 4
    private const val PIXEL_SIZE = 1

    private const val OUTPUT_CLASSES_COUNT = 71
  }
}
```

Тема кваліфікаційної роботи:

Застосування згорткових нейронних мереж для розпізнавання тексту і символів

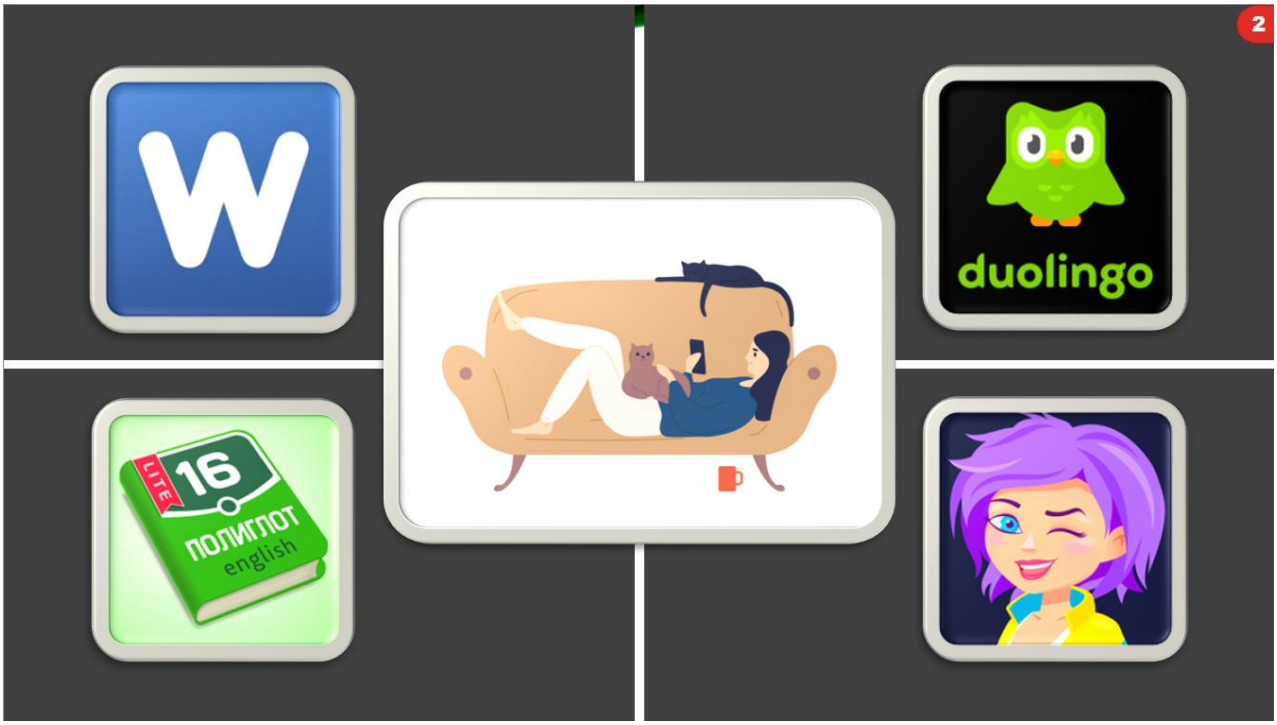
Виконавець:

студент групи КІТм-20-1

Якимаха М. Є.

Науковий керівник:

Сердюк Н. М.



3

4

АКТУАЛЬНІСТЬ РОБОТИ

Актуальність цієї кваліфікаційної роботи має високий рівень через популяризацію азіатської культури, в особливості японської, тому, запотребуваність додатків з вивчання японської мови збільшується. Взагалі, додатків для навчання іноземних мов багато, але таких вузькоспеціалізованих, який було розроблено у цій кваліфікаційній роботі, практично немає.

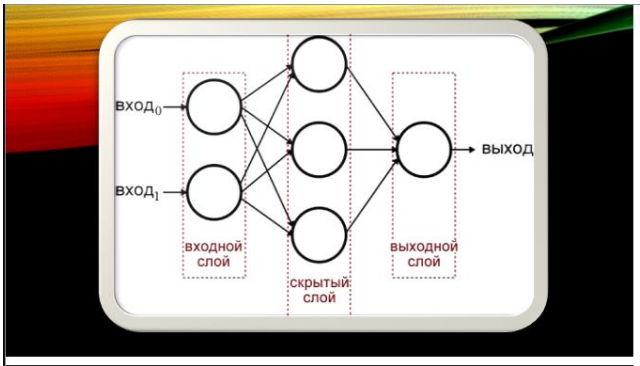
5

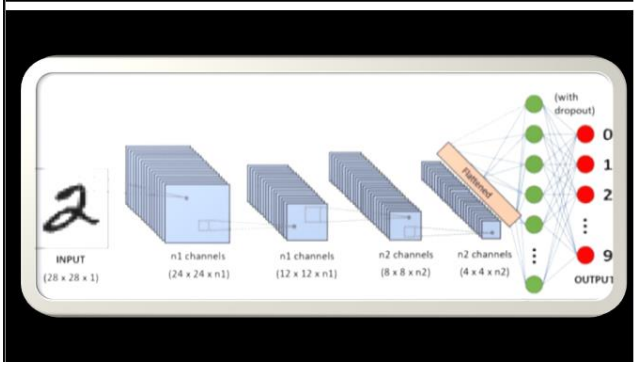
МЕТА РОБОТИ

Метою кваліфікаційної роботи є розробка згорткової нейронної мережі, навчання моделі та розробка додатку для операційної системи Android, для розпізнавання рукописного вводу символу хірагани.

Об'єктом дослідження є використання нейромережі для розпізнавання рукописного символу.

Предметом дослідження є розпізнавання рукописного вводу символу японської слогової азбуки «Хірагана».





6

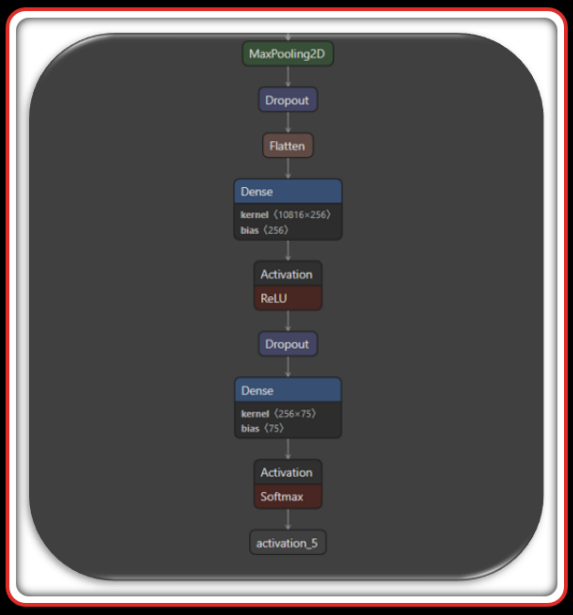
NVIDIA.
CUDA®

7

Згорткова нейронна мережа навчалася за допомоги метода «Навчання з учителем».

Для навчання було використано набір даних, завантажений з ETLcdb. Він має понад 70 тисяч рукописних символів японської складової азбуки хірагана.

8




9
あ
РЕЗУЛЬТАТИ РОБОТИ
НАВЧЕНОЇ МОДЕЛІ


10
お
Prediction Result: お, 5
Confidence: 0.998409
12:48
Hiragana Classifier
お
Prediction Result: お, 5
Confidence: 0.998409
CLEAR
ІНТЕРФЕЙС
ДОДАТКУ

11

ПРИКЛАДИ РОЗПІЗНАВАННЯ СИМВОЛІВ

12

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи, було успішно розроблено згорткову нейронну мережу для навчання моделі, яка розпізнає рукописні символи японської складової азбуки хірагана. Модель була успішно навчена та має точність розпізнавання 99 відсотків, що є позитивним результатом.

Для тестування навченої моделі у реальних умовах було розроблено додаток для мобільної операційної системи Android, а також протестовано модель на правильність розпізнавання рукописних символів складової азбуки хірагана. Результати задовільні, розпізнавання працює без помилок.

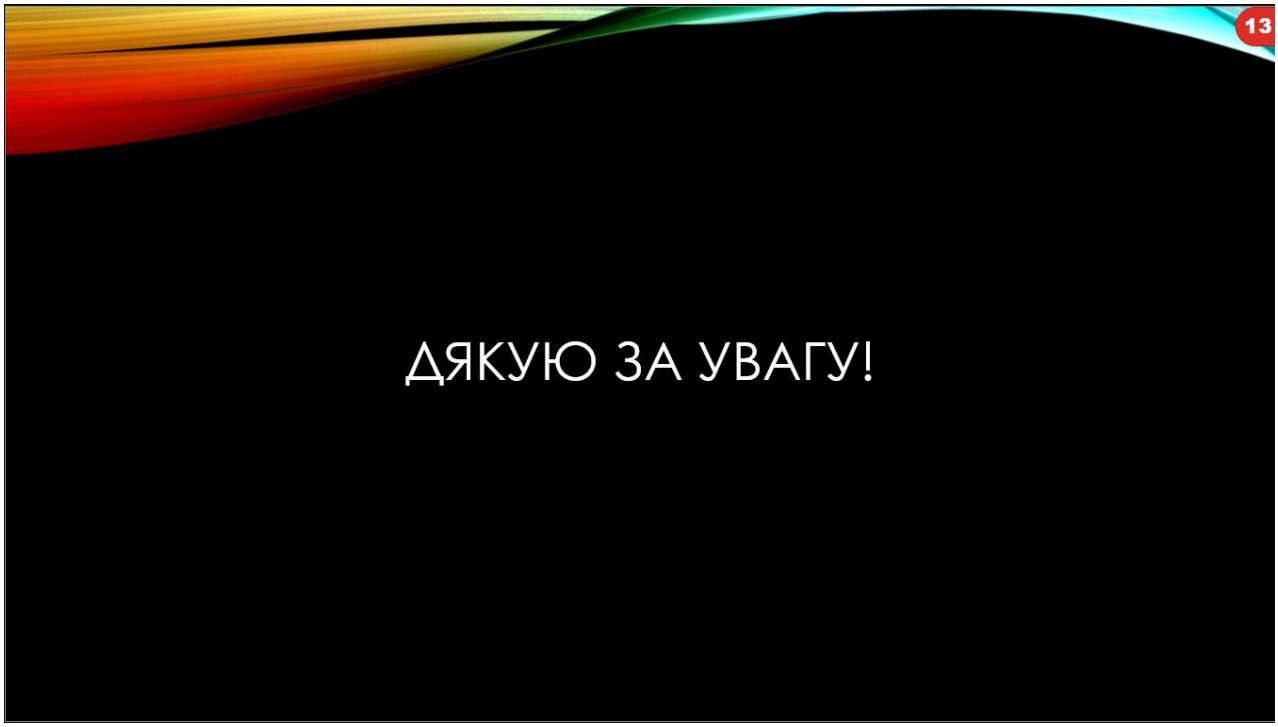
Через те, що на даний момент, популярність додатків з навчання іноземної, в особливості японської мови зростає, практичне використання навченої моделі може знайтись у сфері навчальних додатків підвищеного функціоналу, тобто з можливістю практикування правопису символів та літер.

Публікації здобувача за темою роботи:

1. Якимаха М. Є. Особливості розпізнавання тексту та символів. Радіоелектроніка та молодь у XXI столітті: матеріали 25-го Міжнародного молодіжного форуму. Т. 5. – Харків: ХНУРЕ. 2021. – С. 177 – 178. URL: https://nure.ua/wp-content/uploads/2021/R&M/konferencija-5.pdf.

2. Якимаха М. Є. Використання нейромереж для розпізнавання тексту та символів. Динаміка, рух та розвиток сучасної науки: І Міжнародна студентська наукова конференція. Луцьк. 2021. С. 83 – 84. URL: https://ojs.ukrlogos.in.ua/index.php/liga/issue/view/05.03.2021/468.

Украинская (расширенная)


13
ДЯКУЮ ЗА УВАГУ!