Educational Technologies
Образователни технологии

NEURAL NETWORKS IN A CHARACTER RECOGNITION MOBILE APPLICATION

L.I. Zelenina, L.E. Khaimina, E.S. Khaimin, D.I. Antufiev, I.M. Zashikhina
Northern (Arctic) Federal University named after M.V. Lomonosov (Russia)

Abstract. The article deals with the problem of character recognition using an artificial neural network. Based on the selected data, a model of a convolutional neural network was built, parameters were selected, and the result of training the network was demonstrated. A mobile application with an embedded neural network model has been developed. The process of connecting an API for displaying information on recognized characters has been described. A mobile application with the functionality of recognizing Japanese characters is implemented.

Keywords: convolutional neural network; character recognition; mobile application; application programming interface (API)

Optical Character Recognition Task

Optical Character Recognition (OCR) is the use of technology to recognize printed or handwritten text characters in digital images of documents, such as scanned paper documents (Khaustov, Grigoriev & Spitsyn, 2013).. The basic OCR process involves examining the text of a document and translating characters into code that can be used to process the data. OCR systems consist of a combination of hardware and software that is used to convert physical documents into machine-readable text. Hardware is used to copy or read a text, whereas software usually handles advanced processing. Software can also take advantage of artificial intelligence (AI) to implement more advanced intelligent character recognition (ICR) techniques, such as identifying languages or handwriting styles.

Let's consider the process of character recognition based on neural networks with the further implementation of the results in the form of a mobile application for learning Japanese characters.

Despite the fact that the pronunciation of letters and words in Japanese is simple, the language has a complex spelling system. It consists of two alphabets and hieroglyphs. Learning *kanji* (漢字)) is one of the hardest steps in mastering the Japanese language. In total, there are about ten thousand different hieroglyphs. For everyday use, there is a list of *joyo kanji* (commonly used *kanji*). The list contains 2,136 hieroglyphs. For a language learner, there is no single correct approach to learning hieroglyphs. The task is complicated by the fact that most Japanese characters may be read

differently. The dependence on the word, the position of the hieroglyph in the word, or the context determine the correct reading. Therefore, the developed application will aim at learning hieroglyphs, and not at mere translation. The application can be used, for example, when reading literature, for recognizing inscriptions on the street, etc. Accordingly, the application will be used on mobile devices.

We will use an artificial neural network for character recognition.

Description of the convolutional neural network model

Choosing a dataset for training a neural network

The training data was obtained from the ETL Character Database[1)], which is a collection of images of about 1.2 million handwritten and typewritten digits, symbols, Latin alphabets and Japanese characters collected in 9 datasets (ETL-1 to ETL-9). This database was compiled by the Electrical Engineering Laboratory (now reorganized as the National Institute of Advanced Industrial Science and Technology (AIST)) in collaboration with the Japan Electronics Development Association, universities and other research organizations for character recognition research between 1973 and 1984.

The ETL-9B kit was used for training. The ETL-9B set itself is divided into 5 files (ETL9B_1, ETL9B_2… ETL9B_5). Each file contains 121,440 copies of handwritten hieroglyphs. All 5 datasets were used to train the neural network, i.e. the total number of copies was 607,200. The dataset also contains handwritten copies of Japanese letters from the Hiragana and Katakana alphabet. Since handwritten copies from the alphabet are not hieroglyphs, all the copies from the alphabet were deleted before submitting data to the neural network. The number of different hieroglyphs (classification classes) is 2965. Therefore, the already trained neural network will have to classify 2965 different hieroglyphs of the Japanese language.

Each file from a dataset is a collection of samples in byte format. Figure 1 shows the byte representation of one item.

bytes	# bytes	type
1-2	2	Integer
3-4	2	Binary
5-8	4	ASCII
9-512	504	Packed

Fig. 1. Byte representation of a sample

One copy from the set consists of:
– 2 bytes of integer type – individual instance number,
– 2 bytes of binary type – hieroglyph code in JIS X 0208 (spelling),
– 4 bytes of ASCII string type – standard pronunciation of a hieroglyph,
– 504 bytes – a hieroglyph image. Each image is an array of zeros and ones, where 0 is a black pixel, 1 is a white pixel.
Figure 2 shows the image of one item.

```
test_img = X_images[27424]
test_img = np.reshape(test_img, (64,64))
print(test_img.shape)
plt.imshow(test_img, cmap ="gray")
plt.colorbar()
```

```
(64, 64)

<matplotlib.colorbar.Colorbar at 0x9ff4af0>
```

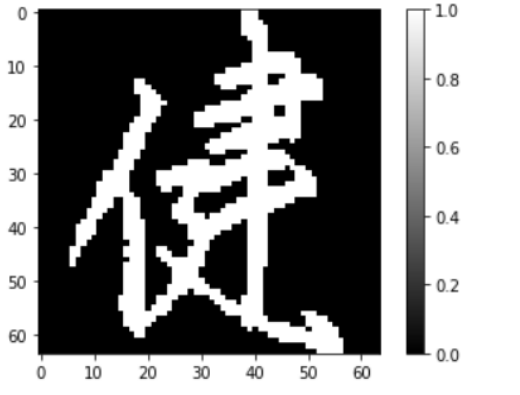

Fig. 2. Image of one sample

Since all items are black and white and represent 0 and 1, they do not need normalization. Therefore, it is advisable to paint the data for supplying a ready-made neural network for prediction black and white and normalize them to the sample from the data (pixelate).

Building a neural network model

To select the type of neural network, it is possible to compare the architecture of the Convolutional Neural Network (CNN) with the architecture of the Recurrent Neural Network (RNN). Dataset of Japanese handwritten characters was used as input, and two tests were carried out for each type of architecture with a different

number of eras and a single sample. Categorical cross-entropy was used as an error function in all tests. Optimization algorithm – Adam (Kosolapov, 2018).

Model	Number of trained parameters	Accuracy	Loss	Accuracy of validity sample
RNN(epochs=4, batch_size = 512)	Tr.Par: 2 534 905 Non-Tr.Par:0	0.8592	0.5087	0.8565
RNN(epochs = 16, batch_size = 256)	Tr.Par: 2 534 905 Non-Tr.Par:0	0.9176	0.2812	0.8828
CNN(epochs = 4, batch_size= 512)	Tr.Par: 1102133 Non-Tr.Par:0	0.9424	0.1883	0.8887
CNN(epochs=16, batch_size = 256)	Tr.Par: 1102133 Non-Tr.Par:0	0.9860	0.0425	0.9251

Fig.3. Outcome of architectures' comparison

According to the data presented, the accuracy is higher for the convolutional neural network (Golikov, 2018). In addition, CNN uses fewer parameters for optimization.

Due to the fact that the average number of items per class (items for one character) was small (≅ 180), convolutional layers and pool layers were added.

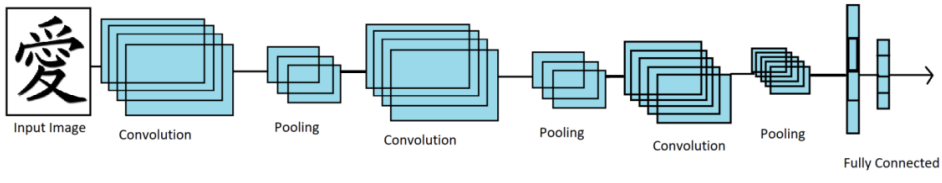

Fig. 4. Convolutional neural network

Model layer architecture:
– x3 Conv_Layers – (32.5); (32.5); (64.5).
– x3 MaxPool_Layers – (2); (2); (2)
– Flatten_Layer
– Dense (2965, "softmax").

The neural network is implemented in Python using the TensorFlow library. Figure 5 demonstrates the creation.

```
model = Sequential(layers=[
   Conv2D(filters=32,kernel_size=5,activation='relu', input_shape=(64,64,1,),-
data_format='channels_last', ),
    MaxPool2D(pool_size=2,),
```

Conv2D(filters=32,kernel_size=5,activation='relu',),
MaxPool2D(pool_size=2,),
Conv2D(filters=64,kernel_size=5,activation='relu',),
MaxPool2D(pool_size=2,),
Flatten()

```
model = Sequential(layers=[
    Conv2D(filters=32,kernel_size=5,activation='relu',
input_shape=(64,64,1,),data_format='channels_last', ),
    MaxPool2D(pool_size=2,),
    Conv2D(filters=32,kernel_size=5,activation='relu',),
    MaxPool2D(pool_size=2,),
    Conv2D(filters=64,kernel_size=5,activation='relu',),
    MaxPool2D(pool_size=2,),
    Flatten()
```

Fig. 5. Sample of listing for creating a convolutional neural network model

This model is a three-layer convolutional neural network, each convolutional layer is connected to the maximum pool layer. The first convolutional layer has 32 filters, with a filter kernel dimension of 5x5. Also, the dimensions of the input image must be supplied to the first convolutional layer. The input image has the dimension (64,64,1) where 64 is the height and width of the image (number of pixels), therefore, each image transmits 4096 pixels to the input of the network, 1 is the color depth of the image. In this set, all images are black and white. In models where color images are submitted for training, parameter 3 will be passed. In the case of color images, each set of 64x64 pixels will display color intensity in RGB, pixel intensity in red, green, and blue colors, respectively. Rectified Linear Unit (ReLU) was selected for all convolutional layers.

The last convolutional layer has an increased number of filters (64), since increasing the number of filters for each subsequent convolutional layer is a good trick to improve the accuracy of the convolutional neural network.

Layer *Flatten*, converts the aspect ratio from a two-dimensional array to one-dimensional. The layer extracts lines of pixels from the image and arranges them in a single row. This layer has no training parameters; it only reformats the data.

Lastly, 256 number of nodes (or neurons) and n-number of nodes are transferred into the fully connected *Dense* neural layers. The second layer returns an array of n-probability scores, giving a total of 1. Each value contains a score indicating the probability of a hieroglyph belonging to one of n classes, where n = 2965.

```
Model: "Seq_01"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_01 (Conv2D)           (None, 60, 60, 32)        832
_________________________________________________________________
maxpool2d_01 (MaxPooling2D)  (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_02 (Conv2D)           (None, 26, 26, 32)        25632
_________________________________________________________________
maxpool2d_02 (MaxPooling2D)  (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_03 (Conv2D)           (None, 9, 9, 64)          51264
_________________________________________________________________
maxpool2d_03 (MaxPooling2D)  (None, 4, 4, 64)          0
_________________________________________________________________
flatten_01 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_01 (Dense)             (None, 256)               262400
_________________________________________________________________
dense_output (Dense)         (None, 2965)              762005
=================================================================
Total params: 1,102,133
Trainable params: 1,102,133
Non-trainable params: 0
```

Fig. 6. Structure of a convolutional neural network's model

Count of all parameters for a convolutional neural network with three convolutional layers = 1102133.

When compiling the model and initializing the final parameters, *adamOptimizer* method, a stochastic optimization method was used. Accuracy was determined as a metric.

Selection of neural network parameters

The *TensorBoard* library was used to select the best model parameters. The visualization of the model indicators allows you to identify the presence of the model's overfitting. To estimate the parameters, the model was run on 9 different variations of the parameters. The conv_layers parameter took the values 1, 2, 3. The epochs parameter took the values 32, 64, 128. The graphs below show the 4 best sets of parameters - [3.32], [3, 128], [3.64], [2, 32], where the first parameter is the number of layers, the second parameter is the dimension of the kernel.

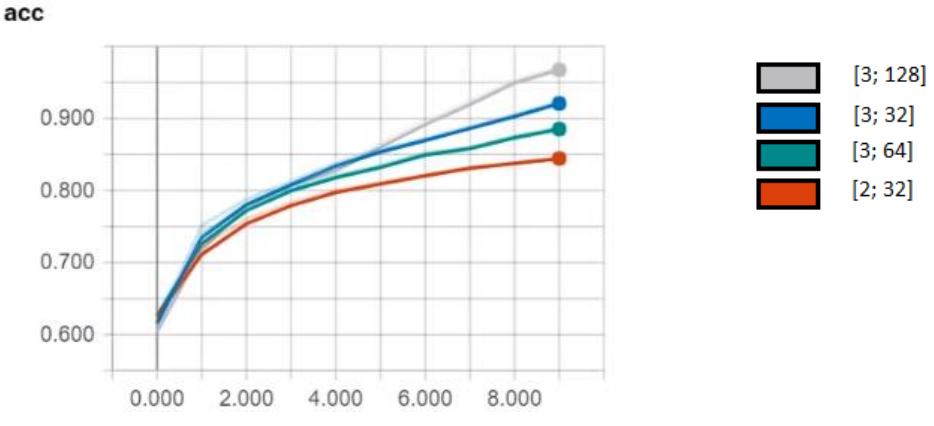

Fig. 7. Model's accuracy

The model [3,128] is shown in gray. In terms of accuracy, the model shows a too high accuracy result. The model with parameters [3, 32] is shown in blue. The accuracy of this model indicates the true accuracy of the non-retrained model more clearly. Based on the four best models described above, we can assume that the most optimal value for the conv_layers parameter can be considered 3.

Figure 8 demonstrates the loss function plot of the top four models.

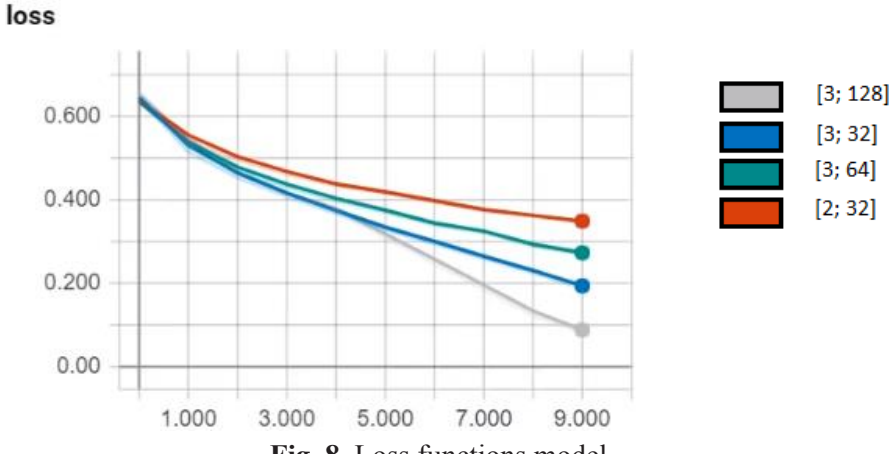

Fig. 8. Loss functions model

The model [3, 128] marked in gray, shows a low level of losses, as well as a too high level of accuracy. This may indicate model's overfitting.

Figure 9 shows a plot of the loss function for the validation set. Analysis and estimation of losses of the data that were not used in the optimization of the neural network parameters can also help in determining the overfitting of the model.

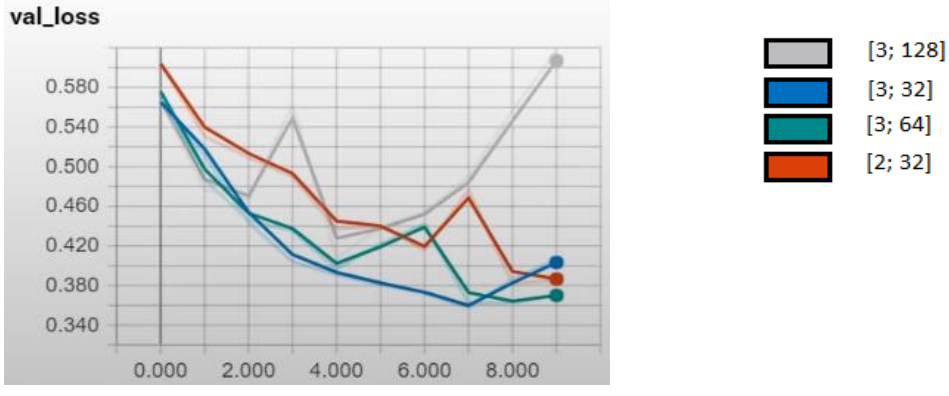

Fig. 9. Plot of the loss function for the validation set

Based on the data in the graph in Figure 9, we can conclude that the model with parameters [3, 128] has undergone retraining.

The model with parameters [3, 32] showed a good result of accuracy both on the entire sample and on the validation sample. For this model, a good result of the loss function was also determined. Also, at the same time, we can conclude that the model is not overfitted, judging by the graph of the loss of the validation sample. Therefore, the final parameters of the model were chosen [3, 32].

Neural network training result

After identifying the most appropriate parameters for the convolutional neural network and directly training the model shows the following results:

– accuracy – 0.9864
– loss (loss) – 0.0420
– val_loss (loss on the validation sample) – 0.1049
– val_accuracy (accuracy on the validation set) – 0.9358

Based on the low value of the loss function on the validation data set, we can say that the model has not undergone retraining.

For the final testing of the model, a part of another dataset consisting of hand-written hieroglyphs was issued for prediction. A model error will be recorded for each hieroglyph.

The result of the check is an object, containing the number of errors in the first prediction of the network for each hieroglyph. Figure 10 shows a part of the check vocabulary.

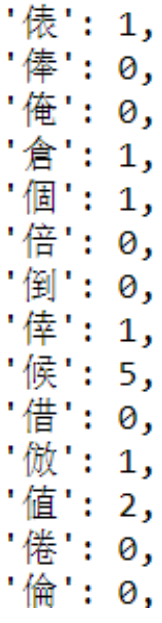

Fig. 10. Assessment of the model's accuracy

To analyze the operation of the constructed neural network, we will determine the number of all errors relative to the size of the full dataset. For this, it is necessary to estimate the number of errors as the first best prediction. The share of erroneous predictions (it is an error if it does not fall into the top 1) is 0.062. For the final test, it is necessary to evaluate the error as the failure to predict the best 6 results. The share of erroneous predictions (it is an error if it does not fall into the top 6) is 0.003.

Based on the above tests, we can say that the convolutional neural network model is trained well. In the application, after the computational work of the neural network, the user will be offered the six best prediction options to choose from.

Mobile application development

The developed application belongs to the class of native applications.

Native apps are specific apps for a specific mobile platform. Since a native app is built for the use on a specific device and its OS, it can use hardware and software specific to the device. Native apps can provide optimized performance and take advantage of the phone's built-in features like GPS, camera, compass, etc.

With the help of a camera, images can be obtained, which are then transmitted to the neural network for processing and obtaining information. For the convenience

of formatting the image received from the camera, you will need to use tools that can directly interact with the camera and allow to scale the resulting image to the desired size for the neural network. It should also be noted that the computational complexity of a trained neural network model is more expedient to use through the API that will communicate with the neural network model.

The *TensorFlow Lite* toolkit is used to embed a neural network model on a mobile device. To implement the task, the Java programming language and development for the Android platform were used.

Using TensorFlow Lite to work with neural networks with mobile applications

After successfully training the neural network, the model is converted to *tflite* format. Thanks to the conversion, the model is only 4.20 MB in size, which is a good advantage for a mobile application. This model helps to meet the requirement of a small application memory footprint. The ready-made converted model must be added to the Android Studio project in the *assets* directory.

Since the model does not take up much space and is not difficult to compute for a mobile device, the optimization of the model through a converter can be neglected.

To work with a neural network, you need to create an object of the *Interpreter* class, the interpreter will configure the API for the interaction of the computing processor of the mobile device with the neural network model.

Interpreter initialization is implemented inside the exception handler to detect errors and check if the interpreter is connected correctly.

After connecting the model and the file with the classes of hieroglyphs, the model can be launched.

Camera app operation

To access the application to the camera of a mobile device, you must register the use of the feature in the XML file AndroidManifest.xml.

For the subsequent use of the camera in the application, it is necessary to initialize requests for permission to use:

– cameras,

– saving data (images) in the device memory,

– eading data (images) in the device memory.

To verify that all permissions to use the camera have been successfully obtained, you need to check the permission code that is sent from the code block to initialize the camera permission requests.

If the user has not received all the permissions, they will receive a message stating that the application must be set to use the designated functions. After the notification, the application stops its functioning.

After obtaining the permission to use the camera, the user must activate the event that facilitates the camera's photo mode.

The application does not use any additional settings at the camera's launch. The standard camera of the device is launched (cameras from the built-in *Camera* application on Android). An example of a camera display is shown by Figure 11.

Fig. 11. Camera Activity

The launched camera activity is the same as the built-in Android camera. Photos taken through the facilitated camera activity are saved in the memory of the mobile device. After a snapshot, the application immediately transfers the last captured image to the *Crop* activity.

Android Crop is an Android library project that provides a simple image cropping action based on the code from AOSP. The library allows you to resize the image to a square size. This is done due to the fact that the neural network model accepts images with a size of 64x64 pixels. A square image is easier to scan to the dimensions described above. A non-square image is almost impossible to resize up to 64x64 without negative effects.

The parameters of the previously initialized requests for permissions to write and read data from the device's native camera are passed to the parameters of the image formatting activity. In the activity, the user needs to scale the

square so that the image includes only the hieroglyph. Figure 12 demonstrates this type of activity.

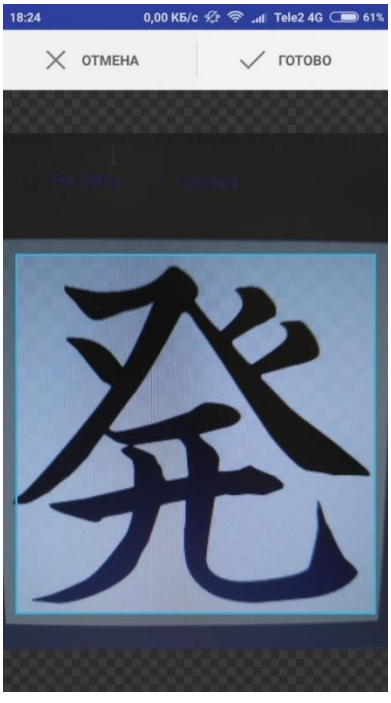

Fig. 12. Formatting activity

In this activity, the user can change the size of the square or the position of the photo so that the image with the hieroglyph only is transmitted to the neural network.

Before transmitting the image to the neural network model, the image must be processed and translated first into a number format, then into a byte format.

After the converting of the image to byte format, the filled object is transferred to the neural network model for prediction.

Neural network model data feed and prediction

When the user clicks the *Define* button (Figure 13), the function of launching the neural network model and the function of showing all the best predictions are launched. For the top three predictions, the probability of the prediction is also indicated. The probability display helps to determine the correctness of the neural network model operation. If the first value is small (<25%), then most likely an error has occurred in determining the background, or the image is subject to noise, or the provided image is low quality.

To make the interface more user-friendly, the top 6 predictions of the neural network model are shown as buttons. Each button has a character from the best selection written on it. The interface for classification of hieroglyphs is shown in Figure 13.

Fig. 13. Hieroglyphs' classification interface

By clicking on the corresponding button, the user goes to the interface for displaying information on hieroglyphs. The user also has the right to choose another suggested hieroglyph if he thinks that the model was mistaken in the first best version or of his own will. Pressing the "back" button returns the user to the starting activity of the application.

API connection for displaying information on hieroglyphs

It was chosen to use a web database in order to display the information on the found hieroglyphs. The embedded database can take up a lot of memory, so it is best to use a remote database connection.

The *Kanjiapi.dev* API (https://kanjiapi.dev/) was chosen as an API that provides data in JSON format for more than 13,000 characters using data from an extensive *kanji* dictionary.

The *Retrofit* library was used to connect to the above API. Retrofit is an HTTP client for working with API. It allows to convert HTTP API into an object of Java language classes[2)].

The most important details of *kanji* were included in the information on the interface for classification of hieroglyphs, namely:

– the meaning of the hieroglyph (keywords),
– reading options (*Kun-Yomi, On-Yomi*),
– the inscription of the hieroglyph,
– important words which contain the hieroglyph (words must be sorted and displayed according to the priority specified in the database).

To receive data through the API, one needs to create a *POJO* (*Plain Old Java Object*) class, which will implement the objects that manifest the fields of *JSON* files and methods for objects' reception. On connecting, the *Retrofit* library will automatically create a class object for the structure of the current JSON file, and then fill its fields with data.

The structure of the received *JSON* files for the hieroglyph's data and the words containing this hieroglyph are shown in Figures 14 and 15.

field	type	description
"kanji":	STRING	*The kanji itself*
"grade":	1..6 | 8..10 | NULL	*The official grade of the kanji (1-6 for Kyōiku kanji, 8 for the remaining Jōyō kanji, 9/10 for Jinmeiyō kanji)*
"stroke_count":	NUMBER	*The number of strokes necessary to write the kanji*
"meanings":	STRING[]	*A list of English meanings associated with the kanji*
"heisig_en":	STRING | NULL	*The Heisig keyword associated with the kanji for English*
"kun_readings":	STRING[]	*A list of kun readings associated with the kanji*
"on_readings":	STRING[]	*A list of on readings associated with the kanji*
"name_readings":	STRING[]	*A list of readings that are only used in names associated with the kanji*
"jlpt":	1..4 | NULL	*The former JLPT test level for the kanji*
"unicode":	STRING	*The Unicode codepoint of the kanji*

Fig. 14. JSON structure of kanji/{kanji} file

For the convenience of working with the data received through the API, the POJO class will be implemented for all fields of the resulting file, even if the fields are not used in the information display activity.

For each field of the POJO class, it is also necessary to create methods to access all the fields of the class (getter_class). Figure 15 shows the structure of a JSON file, which displays all possible words, containing a search character in the {kanji} field.

field	type	description
"meanings":	MEANING[]	*A list of distinct meanings that the entry has*
"variants":	VARIANT[]	*A list of written variations for the entry*

MEANING

field	type	description
"glosses":	STRING[]	*A list of English equivalent terms for the particular meaning*

VARIANT

field	type	description
"written":	STRING	*The written form of the variant*
"pronounced":	STRING	*The pronounced form of the variant (in kana)*
"priorities":	STRING[]	*A list of strings designating frequency lists in which the variant appears*

Fig. 15. JSON structure of the file/ words {kanji}

As you can see from the file structure, each word variant has a priorities field, which is a set of lines that contains information on the popularity of the word in Japanese. To avoid typing too many words on the screen of the mobile device, words will be shown only if the priorities field is not empty, which reduces the list of words to the most important in the language. The data itself is transferred as a set of objects. Therefore, they must be received as a set of objects from the API, in contrast to the information on hieroglyphs. The interface for displaying information on hieroglyphs is shown in Figure 16.

Fig. 16. The page's interface for displaying information on hieroglyphs

As mentioned earlier, the words with priority only are displayed. For each word, the system displays how it is read according to the alphabet. Further, for each word, its translation is displayed. All *TextView* objects for displaying pronunciation (*Kun-Yomi, On-Yomi*) and the field for displayed words are scrollable for readability.

Conclusion

We have considered the process of character recognition based on a convolutional neural network. A mobile application with the functionality of recognizing Japanese characters is implemented.

NOTES

1. ETL Character Database. Accessed online: http://etlcdb.db.aist.go.jp/
2. Retrofit Converter A type-safe HTTP client for Android and Java. Accessed online: https://square.github.io/retrofit/

REFERENCES

Golikov, I. (2018). Convolutional neural network, part 1: structure, topology, activation functions and training set. Accessed online: https://habr.com/ru/post/348000/

Khaustov, P.A., Grigoriev, D.S. & Spitsyn, V.G. (2013). Development of an optical character recognition system based on the combined use of a probabilistic neural network and wavelet transform. Bulletin of the Tomsk Polytechnic University.No. 5 p. 101-105. Accessed online: https://www.elibrary.ru/item.asp?id=20958295/

Kosolapov, K. (2018). Neural networks, fundamental principles of work, diversity and topology. Accessed online: https://habr.com/ru/post/416071/

✉ **Dr. Larisa I. Zelenina, Assoc. Prof.**
Researcher ID: 0000-0002-0155-3139
Department of Applied Mathematics and High Performance Computing
Higher School of Information Technologies and Automated Systems
Northern (Arctic) Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: l.zelenina@narfu.ru

✉ **Dr. Liudmila E. Khaimina, Assoc. Prof.**
Researcher ID: 0000-0003-4552-0440
Department of Applied Informatics and Information Security
Higher School of Information Technologies and Automated Systems
Northern Arctic Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: l.khaimina@narfu.ru

✉ **Evgenii S. Khaimin, Senior Lecturer**
ResearcherID: 0000-0003-0523-3623
Department of Applied Informatics and Information Security
Higher School of Information Technologies and Automated Systems
Northern Arctic Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: e.khaymin@narfu.ru

✉ **Daniil I. Antufiev, Master Student**
Department of Applied Mathematics and High Performance Computing
Higher school of Information Technologies and Automated Systems
Northern Arctic Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: antufjev.d@edu.narfu.ru

✉ **Dr. Inga M. Zashikhina, Assoc. Prof.**
Researcher ID: 0000-0002-8217-2302
Department of Philosophy and Sociology
Higher School of Social Sciences, Humanities and International Communication
Northern (Arctic) Federal University named after M.V. Lomonosov
Arkhangelsk, Russia
E-mail: i.zashikhina@narfu.ru