AN IMPROVED CLASSIFIER AND TRANSLITERATOR OF HAND-WRITTEN PALMYRENE LETTERS TO LATIN

A. Hamplová[*], *D. Franc*[†], *A. Veselý*[‡]

Abstract: This article presents the problem of improving the classifier of hand-written letters from historical alphabets, using letter classification algorithms and transliterating them to Latin. We apply it on Palmyrene alphabet, which is a complex alphabet with letters, some of which are very similar to each other. We created a mobile application for Palmyrene alphabet that is able to transliterate hand-written letters or letters that are given as photograph images. At first, the core of the application was based on MobileNet, but the classification results were not suitable enough. In this article, we suggest an improved, better performing convolutional neural network architecture for hand-written letter classifier used in our mobile application. Our suggested new convolutional neural network architecture shows an improvement in accuracy from 0.6893 to 0.9821 by 142 % for hand-written model in comparison with the original MobileNet. Future plans are to improve the photographic model as well.

Key words: *artificial intelligence, classification, historical alphabets, mobilenet, computer vision*

Received: February 25, 2022 **DOI:** 10.14311/NNW.2022.32.011
Revised and accepted: August 30, 2022

1. Introduction

1.1 Historical alphabet digitization including Palmyrene

Many researches are focused on character recognition of letters from historical alphabets. These include Persian [5], Bangladeshi [3] and cuneiform, which is transliterated by hand [19] and photos of these transliterations are classified.

Until recently, there was no Palmyrene transliteration available. There is a large number of Palmyrene Aramaic memorabilia, which is written in Palmyrene alphabet. It is similar to classic Aramaic with some differences in the alphabet and dialect. This dialect was used in western parts of Syria, and classic Aramaic was

[*]Adéla Hamplová; Czech University of Life Sciences in Prague, PEF KII, Kamýcká 129, CZ-165 00, Praha 6 – Suchdol, Czech Republic, E-mail: hamplova@pef.czu.cz

[†]David Franc – Corresponding author; Czech University of Life Sciences in Prague, PEF KII, Kamýcká 129, CZ-165 00, Praha 6 – Suchdol, Czech Republic, E-mail: francd@pef.czu.cz

[‡]Arnošt Veselý – Corresponding author; Czech University of Life Sciences in Prague, PEF KII, Kamýcká 129, CZ-165 00, Praha 6 – Suchdol, Czech Republic, E-mail: vesely@pef.czu.cz

spoken in the eastern parts. This script was used in the nearest surroundings and inside the Syrian city of Palmyra (also called city of Tadmur) in the years 100–400 A.D.

Translating Palmyrene texts is contributing to the study of ancient art and history, as well as Palmyrene and Biblical studies. The largest anthologies were published in 1996 by Hillers et al. [7] and in 2001 by Taylor et al. [15].

Palmyrene font became part of Unicode in 2010 [1] when the coding for Palmyrene letters was proposed. The alphabet consists of 32 characters in the range 10860–1087F in Unicode [12] (Palmyrene, 2010). Apart from "y", there are only consonants in the script. The alphabet is read from the top right to left corner; words are not divided by a blank space. As for numbers, there are only four characters, which mean 1, 5, 10 (or 100) and 20.

1.2 Image classification on mobile devices

Android applications are developed mostly using the IDE Android Studio and in each version of IDE, a set of standard libraries is added and some of the existing ones are updated. Among standard Android API libraries, image classification is not included. The TensorFlow documentation website [17] recommends using convolutional neural network MobileNet for image classification on mobile devices.

Current research [4] recommends using MobileNet version efficient_lite0. It was introduced by Tan and Le in 2020 [16] and is also presented in Sonawane's paper [14] in comparison with other architectures.

1.3 Android software template and Palmyrene transliterator

In our research we suggest a mobile software tool for automatic character reading of historical alphabets and transliterating them into Latin alphabet. This tool called "Palmyrene Alphabet Transcription" has a potential to help to speed up the process of processing archived but not transliterated and untranslated documents or can be used directly in the field by archeologists.

Our tool is an Android application that can serve as a template for other alphabets as well. The two main use cases of our tool are hand-written letter analysis and letter analysis from photo.

In the first one our tool asks the user to draw a letter on the screen. The drawn image is resized and sent on the input of the convolution neural network (CNN). CNN classifies the image and then the three possible transliterations with the highest confidence score are displayed, see Fig. 1.

The second possibility how to use our tool is using it for transliteration of letters given in photos. Instead of writing the letters by hand, the user takes a photograph of the letter directly from sandstone tablet like in Fig. 2 or other document type that is written in Palmyrene.

The user interface is visible in Fig. 3. There is also a help available, as well as info and alphabet table available in the application.

The target group of users of this tool are archaeologists, who would use the software for faster Palmyrene Aramaic texts transliteration, the secondary focus group are other researchers and people outside the scientific community that could

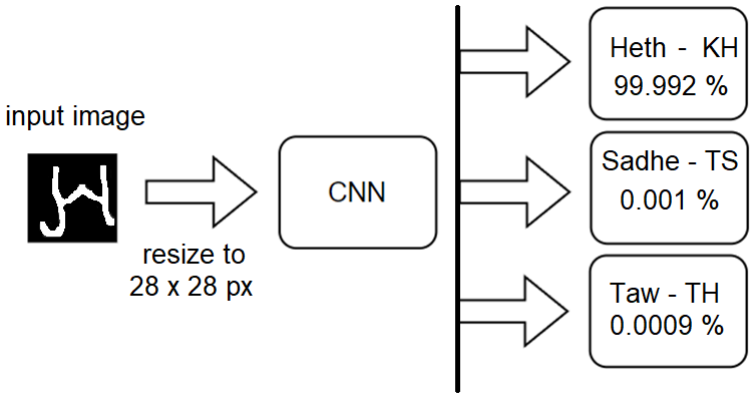

Fig. 1 *In-app character classification.*

Fig. 2 *Example of a sandstone tablet containing Palmyrene script, Inv. 2983/9507, © The Archaeological Museum Of Palmyra.*

use the transliteration for educational purposes. The Android application can also be modified if another historical alphabet model is trained. Therefore, it can be used for further semi-automatic transliteration of any alphabet.

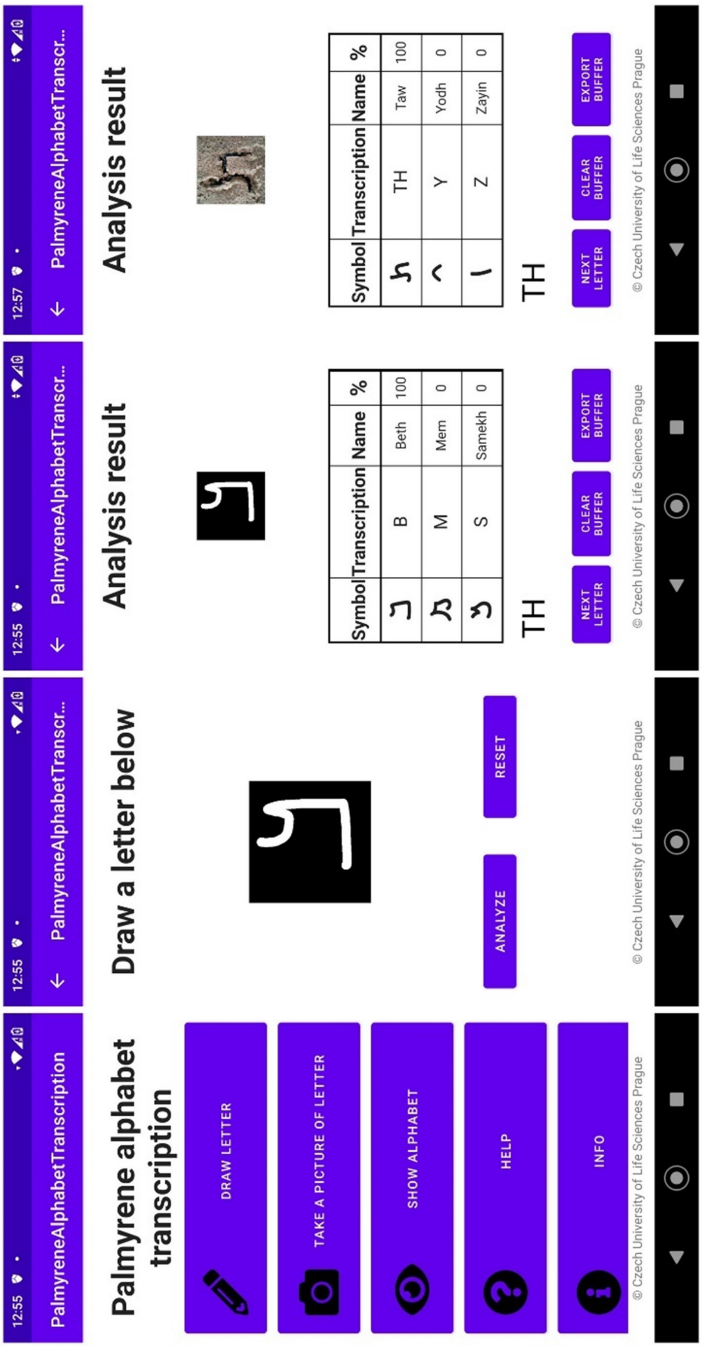

Fig. 3 *"Palmyrene Alphabet Transcription" mobile application user interface.*

2. Objective

Our goal was to design a suitable classifier for transcribing Palmyrene characters into Latin, with special regard to its use in mobile applications. Current research suggests using MobileNet classifier. Therefore, our first step was to verify the possibility of using CNN with the MobileNet architecture and to train it on the Palmyrene alphabet character set. Because the results obtained were not satisfactory, we designed our own CNN architecture, trained it and then we compared the results with the results obtained using MobileNet.

Our results confirmed that it is possible to design CNN architecture of the classifier that gives better results than MobileNet based classifier. It is likely that this classifier would also give good results if trained for transcription of some another similar alphabet.

3. Building the classifier

3.1 Training and validation set

In order to create a dataset of hand-written characters, we modified Axel Thevenot's "Python-Interface-to-Create-Handwrittendataset" tool available at Github [18]. The letters were transcribed from a large number of photographs containing Palmyrene alphabet, such as in Fig. 2. For acquiring these photographs of tablets with Palmyrene inscriptions, we established a cooperation with two museums – Musée du Louvre in Paris [9] as well as Virtual Museum Of Palmyra. [11].

Palmyra alphabet consists of 32 characters, see transcription table in Fig. 4. In our research we considered only 28 characters. We excluded four characters – the numbers 2, 3, 4 and 5. The numbers 2–4 are sequences of the characters 1, and the number 5 looks exactly the same as the letter "ayin". Using graphic tablet, we wrote 56197 letters in total (exactly 2007 samples per each character).

The used system font for Palmyrene is "palmmne". Each character class is saved in a different folder with character name and using keras ImageDataGenerator.flow_from_directory function, the dataset is converted to CNN-readable format. With the generator, the data is split into two subsets – training set contains 80 % data and validation set contains the remaining 20 %. We did not use any data augmentation method.

3.2 MobileNet based architecture

Our first choice of mobile network architecture was picked according to the current recommendations — MobileNet efficient_lite0. This architecture consists of a HubKerasV1V2 layer, Dropout layer to prevent overfitting and an output Dense layer. The final activation function is softmax, as the problem solved is a multiple category classification.

The training of the efficient_lite0 model (both photographic and hand-written) used 80 % images for training and 20 % for validation. For model creation, we used the library "tflite-model-maker" and did not alternate the architecture.

Palmyrene	Latin	Transcription
𐡀	Aleph	‘/A
𐡁	Beth	B
𐡂	Gimmel	G
𐡃	Daleth	D
𐡄	He	H
𐡅	Waw	V
𐡆	Zayin	Z
𐡇	Heth	KH
𐡈	Teth	T
𐡉	Yodh	Y
𐡊	Kaph	K/C
𐡋	Lamedh	L
𐡌	Mem	M
𐡍	Final Nun	M
𐡎	Nun	N
𐡏	Samekh	S
𐡐	Ayin	`/E
𐡑	Pe	P
𐡒	Sadhe	TS
𐡓	Qoph	Q
𐡔	Resh	R
𐡕	Shin	SH
𐡖	Taw	TH
𐡙	left	
𐡚	right	
𐡛	1	
𐡜	2	
𐡝	3	
𐡞	4	
𐡟	5	
𐡠	10, 100	
𐡡	20	

Fig. 4 *Palmyrene characters and transcription to Latin.*

For creating dataset, we call the tflite_model_maker.image_classifier.DataLoader method. By calling this method, input images are resized to 224×224 pixels and loaded into a data generator.

The network is trained calling the function "model.create", which runs training for 5 epochs, with batch size 128 images. The core of the network is not trained, as there are only 38430 trainable parameters and 3451454 non-trainable parameters.

Model summary is specified below.

Model:	"sequential"	
Layer (type)	Output Shape	Param #
Hub_keras_layer_v1v2 (HubKe rasLayerV1V2)	(None, 1280)	3413024
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 30)	38430
Total params:	3451454	
Trainable params:	38430	
Non-trainable params:	3413024	

Tab. I *MobileNet summary.*

The training accuracy was very high even after the first epoch, it reached 0.902, and in the last epoch it reached 0.993. The validation accuracy was 0.677. Model is then saved in .tflite format.

3.3 Design of the custom CNN architecture

To design the network architecture, we conduct experiments with CNN layers. We compare the influence of the number of convolutions in each Convolutional layer, the influence of leaving out pooling layers, compare the difference in performance of AveragePooling and MaxPooling layers and research the influence of the number of repetition of Convolutional/Pooling blocks. We then pick the architecture, which had the highest validation accuracy, lowest validation error and lowest validation loss, and convert it to a mobile version of the model – ".tflite" for testing.

The training is conducted on the graphic card NVIDIA GeForce GTX 970 with memory clock rate 1.1775 GHz, 1664 CUDA cores and memory size 8159 MB.

We created 10 versions of CNN architectures and researched the influence of the combination of layers and numbers of convolutions on a small dataset of handwritten letters (153 per class, image size 28×28). The results are visible in Tab. II, where V is version, Acc_i is accuracy in given epoch i, $Loss_i$ is loss in given epoch i, V_Acc_i is validation accuracy in given epoch i, V_Loss_i is validation loss in given epoch i.

Each architecture version was using alternation of Convolutional layers with specified number of convolutions and Pooling layers, either MaxPooling or AveragePooling. The last three layers were always Flatten and two Dense layers. The versions are described in the following Tab. III, where Conv_i means the number of convolutions in each i-th Convolutional layer.

As visible in Tab. II, the combination of low validation loss and high validation accuracy was present in models using the straight alternation of Convolutional and MaxPooling layers, with 3 or 4 such blocks (mostly versions 1 and 3). Adding another Convolutional / MaxPooling block (version 7) lowered the validation accuracy from 0.548 to 0.4967 in the last epoch and increased the validation loss

V	*Acc*_1	*Loss*_1	*V*_*Acc*_1	*V*_*Loss*_1	*Acc*_15	*Loss*_15	*V*_*Acc*_15	*V*_*Loss*_15
1	0.2992	2.749 00	0.3326	2.6634	0.9688	0.0583	0.5301	2.5247
2	0.7730	0.794 60	0.4330	3.3265	0.9648	0.0511	0.4196	2.7923
3	0.2054	2.871 13	0.2366	2.8368	0.9720	0.0550	0.5480	2.6796
4	0.0649	3.254 80	0.1373	3.0333	0.9389	0.1334	0.4085	5.2541
5	2.8557	0.262 00	0.2334	3.2072	0.9960	0.0758	0.4542	4.2724
6	3.2109	0.082 00	0.1652	3.1675	0.9626	0.0805	0.4743	4.5598
7	3.0733	0.117 90	0.1417	3.2524	0.9628	0.0621	0.4967	4.0713
8	1.4428	0.592 40	0.3571	2.9419	0.9658	0.0499	0.4877	2.9082
9	3.1120	0.123 30	0.1730	3.0210	0.9659	0.6030	0.5580	2.9961
10	3.2842	0.089 40	0.1696	3.0293	0.9652	0.0928	0.5022	3.5259

Tab. II *Influence of CNN layers on network performance.*

V	Conv_1	Conv_2	Conv_3	Conv_4	Conv_5	Pool
1	16	32	64			Max
2	32	64	128			Max
3	16	32	64	128		Max
4	16	32	64	128		Avg
5	32	64	128			Avg
6	32	64	128	256		Avg
7	16	32	64	128	256	Avg
8	16	32	64			Max
9	16	32	64	64		Max
10	16	32	32	32		Max

Tab. III *CNN versions description.*

from 2.6796 to 4.0713. Using AveragePooling layers conducted in lower validation accuracy to 0.4085 in version 4 and almost doubled the validation loss to 5.2541 in comparison to using MaxPooling layers. The best performing architecture overall was version 3.

3.4 Training of the new CNN model

We used the training / validation split equal to 0.8 / 0.2 using the library ImageDataGenerator from tensorflow.keras. The images are resized to 28×28 pixels.

We picked the best performing model with 4 Convolutional layers alternated by 4 MaxPooling layers (version 3). The model summary is visible below.

We trained the network for 15 epochs, with batch size 128, selected optimizer was “adam”. The training results are visible in Tab. V.

Model:	"sequential"	
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	4640
max_pooling2d_1 (MaxPooling2)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_2 (MaxPooling2)	(None, 3, 3, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 128)	73856
max_pooling2d_3 (MaxPooling2)	(None, 1, 1, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 512)	66048
dense_1 (Dense)	(None, 28)	14364
Total params:	177852	
Trainable params:	177852	
Non-trainable params:	0	

Tab. IV *Custom CNN summary.*

Acc_1	*Loss_1*	*V_Acc_1*	*V_Loss_1*	*Acc_15*	*Loss_15*	*V_Acc_15*	*V_Loss_15*
0.861	0.4718	0.9255	0.3092	1	4.66	0.9626	0.3315

Tab. V *Results of final model training.*

4. Evaluation on testing set

The testing dataset is not created in advance. Testing is conducted directly in the mobile application, and the characters need to be written by hand in the "Draw Letter" module. We use 10 samples in each class for testing of both CNN architectures (280 images in total).

4.1 Metrics

For classifier evaluation, the metrics accuracy *acc*, error *err*, recall *r* Eq. (1), precision *p* Eq. (2) and F-score Eq. (3) are used. [2] Accuracy *acc* is the mean of correctly classified characters, error *err* is the mean of incorrectly classified characters. Parameters of recall Eq. (2), precision Eq. (1) and F-score Eq. (3) are evaluated as follows. For each category i we consider binary decision whether character belongs to the category i versus it belongs to any other category $j \neq i$ and we calculate precision p_i, recall r_i and F_i-score.

$$r_i = \sum_{i=1}^{m} \frac{TP_i}{TP_i + FP_i}, \tag{1}$$

$$p_i = \sum_{i=1}^{m} \frac{TP_i}{TP_i + FN_i}, \tag{2}$$

$$F_i = \frac{2 \cdot r_i \cdot p_i}{r_i + p_i}, \tag{3}$$

where
– TP_i is the number of correctly classified objects from category i
– FP_i is the number of characters from the category $j \neq i$ incorrectly classified as being characterss from category i
– FN_i is the number of characters from category i incorrectly classified as being characters from some another category $j \neq i$
– m is the number of categories.

The overall parameters of recall r Eq. (4), precision p Eq. (5) and F score Eq. (6) are then evaluated as arithmetic means.

$$r = \frac{1}{m} \sum_{i=1}^{m} r_i, \tag{4}$$

$$p = \frac{1}{m} \sum_{i=1}^{m} p_i, \tag{5}$$

$$F = \frac{1}{m} \sum_{i=1}^{m} F_i. \tag{6}$$

4.2 Results of MobileNet and custom CNN classifier

The detailed results of MobileNet classifier evaluation on testing set are visible in Tab. VI.

The mean error of this classifier is 0.311, while the mean accuracy reached 0.689.

The least recognized Palmyrene character is “20” and “mem” with only 1 (out of 10) true positive recognition. The character “20” was otherwise classified as “pe” and “waw”, while “mem” was classified as “beth”, “nun” and “pe”. 9 characters – “10”, “aleph”, “beth”, “he”, “nun”, “nun_final”, “pe”, “shin” and “waw” were recognized in each case (10 out of 10).

The classifier was over-oriented for character “pe”, as it had most false positive predictions – 28 other letters were classified as “pe”, as it is visually similar to other characters. The second character with most false positive predictions was “nun” with 18 false classifications and the third one was “nun_final” with 9 false positives.

The detailed results of custom CNN classifier evaluation on testing set are visible in Tab. VII. The mean error of custom CNN classifier is only 0.018, while the accuracy reached 0.982, which is a significant 142 % improvement from the MobileNet classifier.

The least recognized character is “pe” with 8 true positives out of 10, one was classified as “nun” and the other as “yodh”. 24 characters were recognized in all 10 cases out of 10, 3 had one false negative prediction – “gimel”, “resh” and “sadhe”.

There were only 5 characters with false positives – “20”, “daleth”, “heth”, “nun” and “yodh”, each of them had 1 false positive prediction.

class	TP_i	FN_i	FP_i	r_i	p_i	F_i	*err*	*acc*
1	8	2	2	0.8	0.8	0.8		
10	10	0	0	1	1	1		
20	1	9	0	0.1	1	0.182		
aleph	10	0	9	1	0.526	0.690		
ayin	6	4	0	0.6	1	0.75		
beth	10	0	7	1	0.588	0.741		
daleth	5	5	0	0.5	1	0.667		
gimmel	4	6	0	0.4	1	0.571		
he	10	0	0	1	1	1		
heth	9	1	1	0.9	0.9	0.9		
kaph	8	2	2	0.8	0.8	0.8		
lamedh	7	3	0	0.7	1	0.824		
left	2	8	5	0.2	0.286	0.235		
mem	1	9	0	0.1	1	0.182		
nun	10	0	18	1	0.357	0.526		
nun_final	10	0	9	1	0.526	0.690		
pe	10	0	26	1	0.278	0.435		
qoph	7	3	1	0.7	0.875	0.778		
resh	3	7	0	0.3	1	0.462		
right	5	5	0	0.5	1	0.667		
sadhe	4	6	0	0.4	1	0.571		
samekh	3	7	0	0.3	1	0.462		
shin	10	0	2	1	0.833	0.909		
taw	5	5	1	0.5	0.833	0.625		
teth	8	2	0	0.8	1	0.889		
waw	10	0	4	1	0.714	0.833		
yodh	8	2	2	0.8	0.8	0.8		
zayin	7	3	0	0.7	1	0.824		
mean				0.682	0.826	0.672	0.311	0.689

Tab. VI *MobileNet classifier evaluation.*

5. Discussion

The results of the Palmyrene hand-written characters classification were satisfactory, as the accuracy reached 98.21 % instead of 68.93 % with MobileNet efficient_lite0.

The reason behind the false predictions when using MobileNet is the visual similarity of Palmyrene symbols. In sample datasets used for object detection with MobileNet image classifiers, there are many distinct features that makes the classification easier. Such objects like dogs, cats etc. can be stretched, rotated and shifted within the image and still be recognized and classified correctly, however, in case of characters of alphabets, the precise position, shape and rotation of letters matter and can not be altered, because it would change the meaning of the letter, as some letters look like others if rotated or stretched. The demonstration of such

class	TP_i	FN_i	FP_i	r_i	p_i	F_i	err	acc
1	10	0	0	1	1	1		
10	10	0	0	1	1	1		
20	10	0	1	1	0.909	0.952		
aleph	10	0	0	1	1	1		
ayin	10	0	0	1	1	1		
beth	10	0	0	1	1	1		
daleth	10	0	1	1	0.909	0.952		
gimmel	9	1	0	0.9	1	0.947		
he	10	0	0	1	1	1		
heth	10	0	1	1	0.909	0.952		
kaph	10	0	0	1	1	1		
lamedh	10	0	0	1	1	1		
left	10	0	0	1	1	1		
mem	10	0	0	1	1	1		
nun	10	0	1	1	0.909	0.952		
nun_final	10	0	0	1	1	1		
pe	8	2	0	0.8	1	0.889		
qoph	10	0	0	1	1	1		
resh	9	1	0	0.9	1	0.947		
right	10	0	0	1	1	1		
sadhe	9	1	0	0.9	1	0.947		
samekh	10	0	0	1	1	1		
shin	10	0	0	1	1	1		
taw	10	0	0	1	1	1		
teth	10	0	0	1	1	1		
waw	10	0	0	1	1	1		
yodh	10	0	1	1	0.909	0.952		
zayin	10	0	0	1	1	1		
mean				0.982	0.984	0.982	0.018	0.982

Tab. VII *Results of model with* 4×*Conv/MaxPooling network architecture.*

similarity is visible in Fig. 5. The most similar letter to "pe" is "20" and so it had the highest classification error with MobileNet.

When using custom classifier, the improvement is significant (142 % better), as the CNN architecture was tested especially for letter classification and is less prone to error when classifying objects with less distinct features.

Fig. 5 *Visual similarity of character similarity.*

Mara H. conducted the analysis of 3-dimensional scans of tablets with cuneiform signs, however the success rate is not presented in the research. [10] Yamauchi researched hand-written cuneiform characters, but also did not publish classifier results. [19]

Ghosh et al.'s model, that recognized Bangladeshi signs, reached 96.46 % accuracy on MobileNet, which is 2.4 % less than with our upgraded CNN hand-written model. The Bangladeshi script contains 60 letters [3]. Its hand-written were also analysed and reached 90.27 % accuracy. [13]

Our classifier is also comparable with other object recognition tasks, for example Cho Junghwan et al. have researched CT body scans. The dataset contained 4000 very high quality images and they reached 97 % accuracy on GoogLeNet Inception v1 architecture. They also described, how the results declined if less images were used, accordingly [8].

The F-score of 83 % has been reached in the task of great tits and carried food recognition by part of our team. We analysed photos from Smart Nest Boxes. For the model, we used YOLOv3 architecture. The F-measure was lower due to difficult object detection. [6]

Our results of Palmyrene letters recognition were therefore comparable with other author's works. The classifier results were satisfactory, over 70 % as initially stated in the success criteria. With a 98.21 % classification success, the task of hand-written Palmyrene characters classification can be considered resolved.

6. Conclusion

We have explored the architectures suitable for character recognition for mobile use, which is an ever evolving area. Letters and numbers classification is a special image classification case, as, unlike other objects, the images of alphabet characters can not be manipulated rotation-wise, shape-wise and shift-wise. We conducted experiments with convolutional neural network architectures special for character recognition on Android devices, aiming to improve the classification in comparison with MobileNet, and found out, that the network with 4 Convolutional layers alternated by MaxPooling layers has better classification results than other tested networks and trained this network on our data and improved hand-written classifier results by 142 %.

We updated the model in our software tool, which uses artificial intelligence for semi-automation of historical alphabets transliteration and proved its function on Palmyrene Aramaic script. From a general point of view, we can state that if a different model is trained on another alphabet, using the same architecture and mobile application (using different dataset of letters, and with some alternations of in-app texts), this research can serve as a template for other historical script analysis and a foundation of historical optical character recognition (OCR) algorithms.

There is still room for improvement in performance of the photographic model, which is still run on MobileNet and has only 440 images per class in the dataset. We plan to expand the set using keras augmentation and to develop generative adversarial networks. We also plan to create a web application for Palmyrene alphabet recognition, where we will also implement rows recognition and character

segmentation, creating a Palmyrene OCR, aiding researchers with transliterating Palmyrene Aramaic texts in field use and thus contributing to biblical studies. The aim of the next steps of this research is however not just to create a Palmyrene OCR, but to suggest neural network architectures for any historical alphabet character detection, segmentation and classification on mobile and improve the state of art of creating mobile OCR.

Acknowledgement

This article was created with the support of the Internal Grant Agency (IGA) Faculty of Management and Economy, Czech University of Life Sciences in Prague, 2021A0004 – "Reading the characters of Palmyrene alphabet using artificial intelligence tools".

References

[1] EVERSON M. *Proposal for encoding the Palmyrene script in the SMP of the UCS* [online]. UC Berkeley: Department of Linguistics, 2010. Available from: https://escholarship.org/uc/item/27b327h7.

[2] GÉRON A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly, 2019, pp. 88–92.

[3] GHOSH T., ABEDIN M., CHOWDHURY S., TASNIM Z., KARIM T., REZA S., SAIKA S., YOUSUF M. Bangla handwritten character recognition using MobileNet V1 architecture. *Bulletin of Electrical Engineering and Informatics*. 2020, 9(6), pp. 2547–2554, doi: 10.11591/eei.v9i6.2234.

[4] GUPTA D. Mobile Application for Bird Species Identification Using Transfer Learning. *In: 2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*. 2021, pp. 1–6, doi: 10.1109/IICAIET51634.2021.9573796.

[5] HAJIHASHEMI V., ARAB AMERI M.M., ALAVI GHARAHBAGH A., BASTANFARD A. A pattern recognition based Holographic Graph Neuron for Persian alphabet recognition. In: *2020 International Conference on Machine Vision and Image Processing (MVIP)*, 2020, pp. 1–6, doi: 10.1109/MVIP49855.2020.9116913.

[6] HAMPLOVÁ A., PAVLÍČEK J. Object Recognition Tool for "Smart Nest Boxes.". In: *Proceedings of IAC in Budapest 2020. IAC-ETITAI.*, Prague, Czech republic: Czech Institute of Academic Education, 2020, pp. 105–109.

[7] HILLERS D., CUSSINI E. *Palmyrene Aramaic Texts*. Baltimore: Johns Hopkins Univ. Press, 1996.

[8] CHO J., LEE K., SHIN E., CHOY G., DO S. *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?*, 2015. Available from: https://arxiv.org/pdf/1511.06348.pdf

[9] *Le Louvre. Louvre Museum Official Website*. [viewed 2021-12-01]. Available from: https://www.louvre.fr/en.

[10] MARA H., KRÖMKER S., JAKOB S., BREUCKMANN B. GigaMesh and Gilgamesh – 3D Multiscale Integral Invariant Cuneiform Character Extraction. In: *The 11th International Symposium on Virtual Reality, Archaeology and Cultural Heritage. VAST 2010.*, 2010, doi: 10.2312/VAST/VAST10/131-138.

[11] *Palmyra Archaeological Museum*. The Archaeological Museum Of Palmyra, 2021 [viewed 2021-12-01]. Available from: https://virtual-museum-syria.org/palmyra/.

[12] *Palmyrene, Range: 10860–1087F* [pdf]. 2010. Available from: https://www.unicode.org/charts/PDF/U10860.pdf.

[13] SAZAL M.M.R., BISWAS S.K., AMIN M.F., MURASE K. Bangla handwritten character recognition using deep belief network. In: *2013 International Conference on Electrical Information and Communication Technology (EICT)*, 2013, pp. 1–5, doi: 10.1109/EICT.2014.6777907.

[14] SONAWANE P., DROLIA S., SHAMSI S., JAIN B. *Self-Supervised Visual Representation Learning Using Lightweight Architectures*, 2021. Available from: https://arxiv.org/pdf/2110.11160.pdf.

[15] TAYLOR D.G.K. An Annotated Index of Dated Palmyrene Aramaic Texts. *Journal of Semitic Studies*. 2001, 16(2), pp. 203–219, doi: 10.1093/jss/XLVI.2.203.

[16] TAN M., LE Q.V. *Efficientnet: Rethinking model scaling for convolutional neural networks*, 2020. Available from: https://arxiv.org/pdf/1905.11946.pdf

[17] *Image Classification — Tensorflow*. Tensorflow.org, 2021 [viewed 2021-01-14]. Available from: https://www.tensorflow.org/lite/examples/image_classification/overview.

[18] THEVENOT A. Python Interface to Create Handwritten dataset [software]. Available from: https://github.com/AxelThevenot/Python-Interface-to-Create-Handwritten-dataset

[19] YAMAUCHI K., YAMAMOTO H., MORI W. Building A Handwritten Cuneiform Character Imageset. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation. LREC 2018.*, Miyazaki, Japan, 2018. Available from: aclweb.org/anthology/L18-1115.