Weaponizing Unicodes with Deep Learning - Identifying Homoglyphs with Weakly Labeled Data

Perry Deng[§]
Global Cybersecurity Institute
Rochester Institute of Technology
Rochester, USA
perry.deng@mail.rit.edu

Cooper Linsky[§]
Global Cybersecurity Institute
Rochester Institute of Technology
Rochester, USA
cwl4602@rit.edu

Matthew Wright
Global Cybersecurity Institute
Rochester Institute of Technology
Rochester, USA
matthew.wright@rit.edu

***Abstract*—Visually similar characters, or *homoglyphs*, can be used to perform social engineering attacks or to evade spam and plagiarism detectors. It is thus important to understand the capabilities of an attacker to identify homoglyphs – particularly ones that have not been previously spotted – and leverage them in attacks. We investigate a deep-learning model using embedding learning, transfer learning, and augmentation to determine the visual similarity of characters and thereby identify potential homoglyphs. Our approach uniquely takes advantage of weak labels that arise from the fact that most characters are not homoglyphs. Our model drastically outperforms the Normalized Compression Distance approach on pairwise homoglyph identification, for which we achieve an average precision of 0.97. We also present the first attempt at clustering homoglyphs into sets of equivalence classes, which is more efficient than pairwise information for security practitioners to quickly lookup homoglyphs or to normalize confusable string encodings. To measure clustering performance, we propose a metric (mBIOU) building on the classic Intersection-Over-Union (IOU) metric. Our clustering method achieves 0.592 mBIOU, compared to 0.430 for the naive baseline. We also use our model to predict over 8,000 previously unknown homoglyphs, and find good early indications that many of these may be true positives. Source code and list of predicted homoglyphs are uploaded to Github: https://github.com/PerryXDeng/weaponizing_unicode**

***Index Terms*—homoglyphs, unicode, cybersecurity**

I. Introduction

Identical-looking characters, known as homoglyphs, can be used to substitute characters in strings to create visually similar strings. These strings can then be used to trick users into clicking on malicious domains passing as legitimate ones [1], or to evade spam and plagiarism detectors [2]. While the security risks of homoglyphs have been known since the turn of the century [1], the problem has gotten more attention as Unicode becomes the predominant standard for text processing. Unlike ASCII, which contains only 128 characters, the latest Unicode standard encapsulates more than 140 *thousand* characters. Among the Unicode characters, many have been identified to be homoglyphs. This allows scenarios where the entire string of a domain name can be substituted with different Unicode characters and appear visually identical (See Figures 1 and 2). A complete database of all homoglyphs within the Unicode standard would be helpful for us to assess the risks of systems utilizing Unicode. Due to the sheer size of the Unicode standard, however, manual identification of all homoglyphs is infeasible.

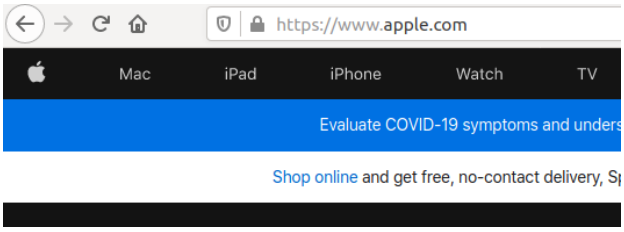

Fig. 1. A screenshot of apple.com, which uses the Latin script

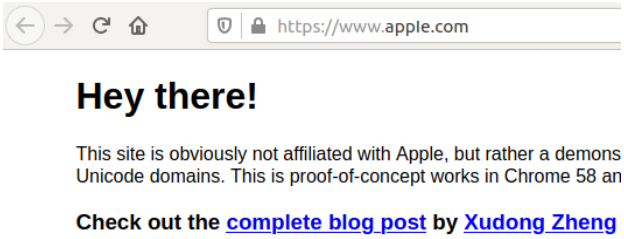

Fig. 2. A screenshot of [3]'s domain, which uses the Cyrillic script

Prior works [4]–[6] on automated homoglyph identification have a few critical shortcomings: 1) they are only used to identify homoglyphs for few characters, or/and 2) they do not cluster homoglyphs into distinct sets, or/and 3) they do not present quantifiable evidence on the accuracy of their approach. We seek to address these shortcomings with empirical evidence on a method based on deep learning, which has performed tremendously well on computer vision problems such as object classification [7] and medical imaging [8]. Unlike many works in computer vision, we do not have a large number of human-labeled examples for each visual class of data, because we do not know the complete sets of homoglyphs. We only know that the overwhelming majority of characters are *not* homoglyphs.

To address this challenge, we propose a novel model that adapts data augmentation, transfer learning, and embedding

This material is based upon work supported by the National Science Foundation under Award No. 1816851.

[§]These authors contributed equally to this work.

learning from similarly challenging deep-learning problems such as facial recognition. We fine-tune our neural network on weakly labeled data by exploiting the fact that, in general, different characters do not look alike. We compare our deep learning model to the approach of Roshabin and Miller [4] on the pairwise identification of homoglyphs. We show that our method significantly outperforms theirs in both accuracy (0.91 vs 0.64) and average precision (0.97 vs 0.75). We also attempt clustering of homoglyphs into distinct sets, and propose to measure the performance and generalizability of our clustering model with mBIOU, a metric based on Intersection Over Union. Our approach achieves a mBIOU of 0.592 between the predicted sets of homoglyphs and the known homoglyphs from Unicode Consortium [9]. In addition, we show that our learned deep embeddings can be used to find many previously unidentified homoglyphs.

In summary, our contributions are as follows:

- We propose a novel deep learning model for identifying homoglyphs that overcomes the lack of training data by leveraging embedding learning and transfer learning, together with data augmentation methods.
- We are the first to attempt clustering of homoglyphs into distinct sets, and propose mBIOU, a way to systematically measure the accuracy of clustered sets.
- We show that our approach outperforms the prior work on the pairwise identification task, with an average precision of 0.97 compared to 0.79. Our approach also achieves 0.592 mBIOU on the clustering task, compared to a naive baseline of 0.430.
- We apply our model to find new homoglyphs, and visualize a random sample of predicted sets. 8,452 unrecorded homoglyphs were predicted with our approach, and visual inspection of our random sample suggests that some of these are likely to be true positives.

II. Background

A. Unicode

Unicode [10] is the predominant technology standard for the encoding, representation, and processing of text. It is maintained by the Unicode Consortium, and is backward-compatible with the legacy ASCII. Unicode defines a space of integers, or *codepoints*, ranging from 0 to 0x10FFFF, which is hexadecimal for slightly more than one million. The entire set of codepoints is called the *code space*. The codepoints can be mapped to *characters*, so that characters can be encoded and processed as unique numbers by computers. As of Unicode Version 13.0 released in March 2020, there are 143,859 characters mapped to codepoints in the standard implemented for public use. For human readability, a codepoint can either be rendered into an image with a font, or printed in hex format with "U+<hex code>". For example, the lowercase first letter of the Latin alphabet can be 'a' or U+0061.

B. Homoglyphs, Homographs, and Confusables

A *homoglyph* is a character that is visually similar to another character. For example, U+0049, or 'I', can appear similar to U+006C, or 'l'. Homoglyphs can be used to substitute characters in strings of one or more characters to form *homographs*, or *confusables* in Unicode terminology. Confusables pose significant security risks. They can fool users into unintended actions, such as malicious domains masquerading as legitimate websites [1]. Confusables can also cause text to slip past mechanical gatekeepers, such as a spam or plagiarism detector [2].

The Unicode Consortium maintains a database [9] of confusables to help mitigate security risks. The incomplete database maps visually similar strings of one or more codepoints to *equivalence classes*, identified by their respective "prototype" string. With this mapping, security practitioners can quickly lookup homoglyphs for a given character or normalize confusable strings into a single encoding [9]. Formally, given the set of possible Unicode strings U^*, and the equivalence relation "is visually similar to", $\equiv$, on U^*, the confusable equivalence class S of a string $\vec{u}$ is:

$$S(\vec{u}) = \{\vec{s} \in U^* | \vec{s} \equiv \vec{u}, \vec{u} \in U^*\} \tag{1}$$

Since there are infinite strings, developing an exhaustive confusable database is impossible. We can instead limit the scope to just single codepoints. More formally, we can modify Equation 1 to have equivalence class (or simply *class*) H of a codepoint c:

$$H(c) = \{h \in U | h \equiv c, c \in U\} \tag{2}$$

where U is the set of all renderable Unicode codepoints. When $|H(c)| \geq 2$, we get *non-trivial sets* of two or more homoglyphs (as each character is always in an equivalence class containing at least itself). With these homoglyph sets, we can form longer confusables through character substitution and assess risks in vulnerable systems.

C. Automated Homoglyph Identification

Homoglyph identification can be understood as the problem of finding all sets of equivalence classes that satisfy Equation 2. An alternative problem formation, used in all previous work, is to identify pairs of homoglyphs rather than sets. Pairs are less useful than sets, because for us to store all the homoglyph associations of all given characters, pairwise lookup would require drastically more storage. To the best of our knowledge, prior work on the automated identification of homoglyphs within Unicode includes:

- Roshanbin and Miller [4] develop an approach based on Normalized Compression Distance (NCD) that identifies *pairs* of homoglyphs from a set of around 6,200 Unicode characters. They claim that their study expands the set of known homoglyphs at the time of their work.
- Ginsberg and Yu [5] aim to identify potential homoglyph pairs by decreasing their granularity and normalizing their sizes and locations. They present empirical results on a select few letters, and mainly focus on the speed of their method. Like Roshanbin and Miller, they did not attempt to map codepoints into equivalence sets.

- Sawabe et al. [6] use optical-character-recognition (OCR) software to produce a dynamic mapping from Unicode characters to ASCII characters. They assesses the OCR homoglyph detection scheme as part of a larger model, evaluating its utility to detect confusable URLs against existing Unicode-ASCII mappings. Although OCR technologies have become more sophisticated and effective in recent years, they are typically designed to recognize a small subset of Unicode. This makes it impossible to identify more equivalence classes than the number of visually distinct characters supported by the OCR software, which is fewer than 128.

No prior works quantify the accuracy of their approach.

D. Few-Shot Learning

Deep-learning approaches typically perform poorly without a lot of data. Recent work has attempted to overcome this by addressing the so-called *few-shot* learning problem, where just a few examples are given for a class. Few-Shot classification or clustering problems can be described as "i-shot, j-way", where i stands for the number of examples given for a class, and j is the number of classes. Many approaches [11] have been proposed to improve the performance of deep-learning methods in few-shot learning, with the most promising ones being data augmentation, embedding learning, and multitask/transfer learning. We leverage some of these techniques in our work.

III. Deep Learning Approach

A. Learning an Embedding Function

Classifying visually similar glyphs into distinct sets can be considered as a computer vision task. There has been prior work in computer vision tackling similar problems, such as using Convolutional Neural Networks (CNNs) for 1-shot, 20-way classification on the Omniglot dataset [12]. The Omniglot dataset is similar to our problem, in that the problem is glyph classification. It contains 1623 distinct characters, and during testing, the model is asked to predict between 20 character classes. There are, however, many more than 20 equivalence classes (Equation 2) in Unicode. For instance, if (conservatively) 80% of codepoints do not look similar, then we would have more than 115,000 classes. In that aspect, our problem is more like mass scale facial recognition than 20-way character classification.

To tackle this, we utilize several techniques from the few-shot learning problem domain, including data augmentation, transfer learning, and embedding learning.

Data Augmentation: One issue with our data is not knowing the equivalence class membership of a character in general. In other words, we lack labels for nearly all characters. While there are public databases of known homoglyphs, there is no guarantee that they are representative of the entire codespace. The confusable database maintained by Unicode Consortium [9], for example, underrepresents Chinese-Japanese-Korean characters in their database, leading us to suspect that most homoglyphs are still unrecorded. In addition, training on the few known class labels makes it harder to assess the model's ability to generalize. We therefore forego the use of known homoglyph labels for training. Instead, we generate *weak labels* by exploiting the fact that, in general, characters are not homoglyphs, and belong to their own trivial equivalence classes. This allows us to sample characters from different classes by random uniform choice, with reasonable reliability.

Another issue is that we only have a few example images for each codepoint, from various fonts' implementations of the same character. This means that we only have a few datapoints for each weakly labeled class. To alleviate this problem, we augment our data by randomizing the location of a glyph within an image and performing random affine transformations on images during training.

Embedding Learning: Embedding learning is a type of modeling where, instead of directly learning a classification or regression model, we learn an *embedding function* that outputs relevant latent variables. Embedding learning excels in unsupervised and semi-supervised few-shot learning problems [11]. Neural network architectures are good candidates for the embedding functions, because they can approximate highly non-linear functions.

To train our embedding function, we utilize a modified version of the Triplet Loss function from Facenet [13], a work on facial recognition. In Triplet Loss, a triplet is consisted of an *anchor* datapoint, a *positive* datapoint from the same class as the anchor, and a *negative* datapoint from a different class. Having computed the embeddings for all three samples, Triple Loss is optimized when the distance between the anchor and positive embeddings is low compared to the distance between the anchor and negative embeddings.

To produce our triplets, we use the weak labels to generate an anchor and a positive sample from the same codepoint, with a negative sample from a different codepoint. For computing Triplet Loss, though a Euclidean distance between embeddings might be used, we instead use one minus the cosine similarity. We also did not implement the complicated triplet mining algorithm proposed by Schroff et al. [13].

Transfer Learning: For our embedding function, we use EfficientNet [7] without its softmax classification layer. It is a CNN, and the feature maps of its penultimate layer are flattened to become our embedding vectors. EfficientNet is pretrained on ImageNet, and has achieved state-of-the-art transfer performance in various datasets [7]. We apply a regularization loss by penalizing the L_2 distance between our learned weights and the pretrained weights.

B. Clustering Homoglyphs into Equivalence Classes

Given the learned embedding function, we can separate the embeddings into equivalence classes by assigning them to distinct clusters. While many methods exist for unsupervised clustering problems, most are designed with a fixed number of clusters in mind. They are thus not suited for our problem, where a huge unknown number of single datapoint clusters (non-homoglyph characters) exist. We propose an ad-hoc heuristic, where each character is iterated through, assigned to

an existing cluster where all characters have a similarity with the candidate character greater than a particular threshold, or a new cluster of its own. Afterwards, we iterate through all proposed clusters C_i in a list, comparing them to the merge candidates, $C_{k,k>i}$, that are clusters further down the list. We calculate the mean and variance of cosine similarities between the characters in each C_k and characters in C_i, and merge C_k with C_i if certain thresholds apply to the mean and variance of the cosine similarities.

C. Rendering the Codepoints

To render U, we use a collection of Google NoTo fonts, and Windows and MacOS default fonts. For consistency, we convert each font to the TrueType format, the most common format in the set. Each TrueType font contains a list of supported Unicode codepoints and a graphical representation of how they are displayed. Although there are 143,859 characters mapped to codepoints by the Unicode Consortium, a codepoint can only be utilized if it is renderable by at least one font. For a codepoint to be renderable by a font, it must meet the following criteria:

- The character must be on the font's list of supported characters.
- The font does not render the character as a replacement character, which occurs when a font is unable to represent a Unicode character graphically.

116,294 characters meet these criteria and are used for training and testing. During training, we treat multiple font implementations of the same character as multiple datapoints of the same weakly labeled class, and the specific font to render a character is selected at random. In our experiments for model assessment and comparison, we follow prior work and treat each character as a single datapoint. As Roshanbin and Miller point out [4], ideally, the same font would render every codepoint. However, there is no such font that supports rendering every Unicode character. We therefore use the minimum possible number of fonts, as [4] does, in our experiments.

IV. Experiments

A. Metrics: Clustering

To assess the efficacy of our approach at clustering Unicode codepoints into homoglyph equivalence classes, we conduct several experiments on our trained model to show our approach predicts equivalence classes that are close to ground truth. Since the ground truth equivalence classes for Equation 2 are unknown over the set U, it is impossible for us to directly measure the fit of predicted equivalence classes. Luckily, Unicode Consortium [9] maintains a database of known confusable strings. About 5000 of these confusable strings are length one strings, or homoglyph characters, which are useful to compare our predictions against. These homoglyphs are grouped into distinct sets, which satisfy our definition of equivalence classes. 4666 of these homoglyphs can be rendered by the fonts built into our operating systems. This gives us a total of 1313 non-trivial equivalence classes. For the rest of this section, we refer to these as the ground truth equivalence classes on U'. Formally, let's define U' as the set of 4666 known homoglyphs from Unicode Consortium, we can then substitute U with U' in Equation 2, which yields 1313 ground truth equivalence classes satisfying the definition:

$$H'(c) = \{h \in U' | h \equiv c, c \in U'\} \tag{3}$$

Most of the 1313 equivalence classes have a cardinality no more than 10, with the largest being 71 (Table I). We refer to these ground truth individual equivalence classes as $H'_1, H'_2, \ldots, H'_{1313}$. Likewise, we refer to our each of our k predicted equivalence classes as $H^p_1, H^p_2, \ldots, H^p_k$, where k can be any number of equivalence classes predicted, and p simply indicates "predicted."

TABLE I
Distribution of Equivalence Class Cardinality in U'

Cardinality	Number of Equivalence Classes
2 - 10	1237
11 - 20	45
21 - 30	23
31 - 40	5
41 - 50	1
51 - 60	0
61 - 70	1
71 - 80	1

Therefore, we estimate the fit by measuring the distance between our equivalence class predictions (predicted sets) and the ground truth sets over U' rather than U. The metric we use is *mean Best Intersection Over Union* (mBIOU). Intersection Over Union is a way to measure the distance between two sets, commonly used in object detection problems. mBIOU first finds the best matching between the predicted sets and ground truth sets. For example, if there is a predicted set of characters that look like the letter 'a', it will be matched with a similar set of ground truth characters, if such a set exists. Given this matching, mBIOU computes the average IOU between the matched pairs.

Formally, we define

$$mBIOU = \frac{1}{n} \times \sum_{i=1}^{n} \frac{|H'_i \cap H_i^{best}|}{|H'_i \cup H_i^{best}|} \tag{4}$$

where $n = 1313$ is the total number of ground truth equivalence classes in U', and

$$\begin{aligned} H_i^{best} &= \arg\max_{H_j^p} |H'_i \cap H_j^p| \\ &\text{s.t.} \quad 1 \leq j \leq k \end{aligned} \tag{5}$$

To show that the proposed mBIOU metric is a good measure of the fitness of our equivalence class predictions, let us consider its behavior in several thought experiments:

- *The predicted sets align perfectly with the ground truth*: mBIOU = 1.0 because the intersections are exactly the same as unions.

- *The predicted sets have only one character of overlap with the ground truth, the smallest possible intersection*: mBIOU will be low, because the intersection cardinality will not exceed one, which suppresses IOU for non-trivial equivalence classes.
- *The predicted sets contain many false positives for any class, i.e., low precision*: mBIOU would be low due to having relatively large unions.
- *The predicted sets contain few true positives for any class, i.e., low recall*: mBIOU would be low due to having relatively small intersections.

From these scenarios, we can see that mBIOU is a reasonably good measure of how close our predictions over U' are to ground truth.

It is important to note, however, that U' has some different properties than U, which is the entire set of more than 100,000 renderable Unicode codepoints. Critically, assuming that the distribution of cardinality for ground truth equivalence classes remains similar, the number of negatives for each of the classes will be much, much larger. It is therefore crucial to have high precision. To assess the ability of our proposed approach on U, we add in new random codepoints to U', and obtain U'' and U''' for 1000 and 3000 additions, respectively.

Importantly, the predicted equivalence classes are measured against $H'_1, H'_2, \ldots, H'_{1313}$, which do not include the new random addition of characters. This is because: 1) we do not know which equivalence classes the new additions belong to; 2) we know that they very likely do not belong to the known equivalence classes on U', otherwise they would have been identified easily by humans who already found the original classes; 3) if we modify our ground truth classes to include these trivial sets of non homoglyphs, then the mBIOU can be inflated if it is good at clustering non-homoglyphs to their own sets, which will not give us a meaningful way of telling whether our model can generalize to U with high precision. We also directly provide a measure of precision, *mean best precision* (mBP), defined as

$$mBP = \frac{1}{n} \times \sum_{i=1}^{n} \frac{|H'_i \cap H_i^{best}|}{|H'_i \cup H_i^{best}|} \tag{6}$$

where $H_{best,i}$ is defined in Equation 5. Assuming that these random additions of characters do not belong to any of the ground truth equivalence classes on U', for our model to generalize to U, we expect mBIOU and mBP to not fall off meaningfully on these random additions of negatives. We also provide mBIOU and mBP for a *naive baseline* approach, where each codepoint belongs to its own predicted equivalence class of size one.

B. Metrics: Comparing to Prior Work

We also compare our approach to prior work, which focused on identifying homoglyph pairs rather than clusters of equivalence classes. Note that Ginsberg and Yu's work [5] is an attempt at *rapid* homoglyph detection, and presents little evidence that their approach is also accurate. Sawabe et al.'s OCR approach [6] can only detect homoglyphs for ASCII characters, and thus it is not applicable to more than 99% of Unicode characters. As such we solely focus on Roshanbin and Miller's work [4] in our comparison. To this end, we sample pairs of characters from U', which is labeled data. We then calculate the accuracy, precision-recall-curve (PR curve), and average precision for our approach and their approach. To be more specific, let us define all possible homoglyph pairs from U' as the set

$$P = \{(a, b) | a \in U', b \in U', a \neq b, a \equiv b\} \tag{7}$$

and all possible non-homoglyph pairs from U' as the set

$$N = \{(a, b) | a \in U', b \in U', a \neq b, a \not\equiv b\} \tag{8}$$

In one trial of n pairs (we use $n = 2000$), we sample with replacement $n/2$ pairs from P and $n/2$ pairs from N. For each pair, we sample characters a and b uniformly at random, where $a \equiv b$ for a P datapoint and $a \not\equiv b$ for an N datapoint. Then we obtain the cosine similarity between a and b on our learned embedding of the sampled pair. We also calculate the NCD with the LZMA compressor [4]. This allows us to train two linear kernel support vector machines, one for the cosine similarities and one for the Normalized Compression Distance, to have large margin classifications for whether a pair belongs to P (positive) or N (negative). We use the compression function in the LZMA library in Python 3 with default parameters. We train the SVMs, calculate the accuracy and average precision, as well as plot the precision-recall curve using the sklearn library.

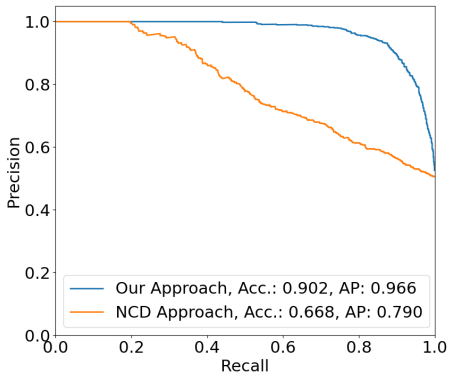

Fig. 3. PR curve, accuracy (Acc.), and average precision (AP) of our method and NCD on identifying homoglyph and non-homoglyph pairs (n = 2000)

C. Results

1) Clustering Homoglyphs: Table II contains the clustering results. It might be confusing to see that the baseline approach achieves precisely the same mBIOU and mBP across all datasets, even the ones augmented with negative examples. This is because the ground truth sets are the same across three datasets, and thus for each H'_i, we get a H_i^{best} that is composed of one character from H'_i. This results in identical mBIOU and mBP, across all three datasets. Apart from this observation, we see that our approach is better than the baseline by a

TABLE II
MBIOU AND MBP OF PREDICTED HOMOGLYPH SETS

Data	U'	U''	U'''
mBIOU	0.592	0.580	0.566
mBP	0.697	0.696	0.696
mBIOU (Baseline)	0.430	0.430	0.430
mBP (Baseline)	0.430	0.430	0.430

significant margin. We also see some drop in mBIOU as more negative examples are added, without corresponding drop in mBP, suggesting that the approach needs work on recall before it can generalize well to U.

2) Comparing to Prior Work: With an overall accuracy of 0.90 and average precision of 0.97, our method greatly outperforms Roshanbin and Miller's method [4] (Figure 3) for identifying homoglyph pairs. Given the same number of false positives, our method can potentially identify many more homoglyphs. This also hints that, if applied to clustering homoglyphs into equivalence classes, NCD will likely fare worse than our deep-learning approach.

V. FINDING PREVIOUSLY UNKNOWN HOMOGLYPHS

We also test whether our approach can find previously unknown homoglyphs. To do this, we utilize the pairwise character cosine similarity matrix A of size N^2, where $N = 116,294$ is the total number of renderable codepoints. We apply a handpicked threshold α and look for rows in A with more than one element greater than the threshold. Note that each row will always have at least one element passing the threshold due to self-comparison (a cosine similarity of 1.0). We sampled thresholds in the range $-1 < \alpha < 1$ due to the range of values for cosine similarity. Using an arbitrary value of $\alpha = 0.93$, our model predicts 8452 homoglyphs previously unrecorded by Unicode Consortium, which is *more than two times* the recorded number. We show a random sample of four such sets of predicted homoglyphs below (Figure 4). Due to the relevance of Latin Scripts in WWW use today, we also present a set of previously unidentified Latin-confusable homoglyphs (Figure 5). While some of these are false positives, many appear to be genuine homoglyphs. On the other hand, it is arguable whether the characters look visually confusing to native users of the language scripts. The full list of predicted homoglyphs is on Github.

VI. DISCUSSION

Outperforming prior work on homoglyph identification and achieving promising results on clustering Unicode homoglyphs, our work shows that deep learning can effectively generate embeddings for Unicode characters, which can not only be used by attackers to find homoglyphs (as in our case), but also possibly by defenders to detect phishing attacks using homograph strings (see [14] for an example of defense). Further research is still needed, because we did not address composite characters of multiple Unicode codepoints, or composite homoglyphs of multiple characters. Our approach has a far from perfect mBIOU, and can likely be improved by utilizing existing state of the art algorithms that cluster embeddings into an unknown number of classes (e.g., hierarchical clustering). While we predicted 8452 homoglyphs, domain experts or surveys are needed to determine the prediction quality.

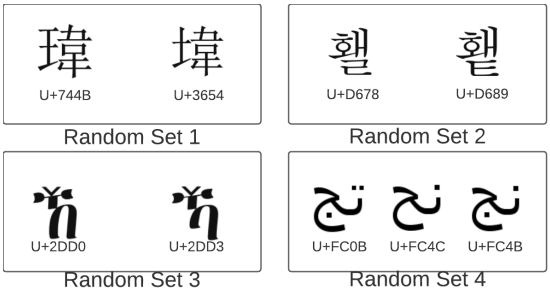

Fig. 4. Four random sets of predicted homoglyphs previously unidentified

Fig. 5. A latin-confusable set of predicted homoglyphs previously unidentified

REFERENCES

[1] E. Gabrilovich and A. Gontmakher, "The homograph attack," *Communications of the ACM*, vol. 45, no. 2, 2002.
[2] D. Weber-Wulff, C. Möller, J. Touras, E. Zincke, and H. Berlin, "Plagiarism detection software test 2013," *Abgerufen am*, vol. 12, 2013.
[3] X. Zheng, "Phishing with Unicode domains – Xudong Zheng," 2017. [Online]. Available: https://www.xudongz.com/blog/2017/idn-phishing/?_ga=2.53371112.1302505681.1542677803-1987638994.1542677803
[4] N. Roshanbin and J. Miller, "Finding homoglyphs – a step towards detecting Unicode-based visual spoofing attacks," in *WISE*, 2011.
[5] A. Ginsberg and C. Yu, "Rapid homoglyph prediction and detection," in *ICDIS*, 2018.
[6] Y. Sawabe, D. Chiba, M. Akiyama, and S. Goto, "Detection method of homograph internationalized domain names with OCR," *Journal of Information Processing*, vol. 27, 2019.
[7] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.
[8] E. D. Pisano, "AI shows promise for breast cancer screening," 2020.
[9] M. Davis and M. Suignard, "UTS #39: Unicode security mechanisms," 2020. [Online]. Available: https://unicode.org/reports/tr39/#confusables
[10] "About the Unicode® standard," 2017. [Online]. Available: https://www.unicode.org/standard/standard.html
[11] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, 2020.
[12] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, 2015. [Online]. Available: https://science.sciencemag.org/content/350/6266/1332
[13] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *CVPR*, 2015.
[14] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Detecting homoglyph attacks with a siamese neural network," in *DLS*, 2018.