

A Multi-modal Neural Embeddings Approach for Detecting Mobile Counterfeit Apps

Jathushan Rajasegaran
Data61, CSIRO
Sydney, Australia
brjathu@gmail.com

Naveen Karunanayake
Data61, CSIRO
Sydney, Australia
naveenharshitha95@gmail.com

Ashanie Gunathillake
The University of Sydney
Sydney, Australia
ashanie87@gmail.com

Suranga Seneviratne
The University of Sydney
Sydney, Australia
suranga.seneviratne@sydney.edu.au

Guillaume Jourjon
Data61, CSIRO
Sydney, Australia
guillaume.jourjon@data61.csiro.au

ABSTRACT

Counterfeit apps impersonate existing popular apps in attempts to misguide users. Many counterfeits can be identified once installed, however even a tech-savvy user may struggle to detect them before installation. In this paper, we propose a novel approach of combining *content embeddings* and *style embeddings* generated from pre-trained convolutional neural networks to detect counterfeit apps. We present an analysis of approximately 1.2 million apps from Google Play Store and identify a set of potential counterfeits for top-10,000 apps. Under conservative assumptions, we were able to find 2,040 potential counterfeits that contain malware in a set of 49,608 apps that showed high similarity to one of the top-10,000 popular apps in Google Play Store. We also find 1,565 potential counterfeits asking for at least five additional dangerous permissions than the original app and 1,407 potential counterfeits having at least five extra third party advertisement libraries.

CCS CONCEPTS

• **Networks** → **Mobile and wireless security.**

KEYWORDS

Mobile Apps; App Security; Security; Fraud Detection

ACM Reference Format:

Jathushan Rajasegaran, Naveen Karunanayake, Ashanie Gunathillake, Suranga Seneviratne, and Guillaume Jourjon. 2019. A Multi-modal Neural Embeddings Approach for Detecting Mobile Counterfeit Apps. In *Proceedings of the 2019 World Wide Web Conference (WWW '19)*, May 13–17, 2019, San Francisco, CA, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3308558.3313427>

1 INTRODUCTION

Availability of third party apps is one of the major reasons behind the wide adoption of smartphones. The two most popular app markets, Google Play Store and Apple App Store, hosted approximately 3.5 million and 2.1 million apps by early 2018 [16, 24]. Handling

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

WWW '19, May 13–17, 2019, San Francisco, CA, USA

© 2019 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License.

ACM ISBN 978-1-4503-6674-8/19/05.

<https://doi.org/10.1145/3308558.3313427>

such large numbers of apps is challenging for app market operators since there is always a trade-off between how much scrutiny is put into checking apps and encouraging developers by providing fast time-to-market. As a result, problematic apps of various kinds including malware have made it into the app markets [42, 56].

One category of such problematic apps is *counterfeits* (i.e. apps that attempt to impersonate popular apps). Reasons behind app impersonations include harvesting user credentials, increasing advertising revenue, and spreading malware. Many popular apps such as *Netflix*, *IFTTT*, *Angry Birds*, and *banking apps* have been reported to be affected by counterfeits [8, 25, 39, 50]. In Figure 1, we show an example counterfeit named *Temple Piggy*¹ which shows a high visual similarity to the popular arcade game *Temple Run*.²



a) Original (Temple Run)



b) Counterfeit (Temple Piggy)

Figure 1: An example counterfeit for the game Temple Run

In this paper, we propose a neural embedding-based approach to identify counterfeit apps from a large corpus of apps. We leverage the recent advances in Convolutional Neural Networks (CNNs) to generate feature embeddings from given images using pre-trained models such as VGGNet [45]. In contrast to commonly used *content embeddings* generated from fully connected layers before the last soft-max layer, we show that combining *content embeddings* with *style embeddings* generated from the Gram matrix of convolutional layer feature maps achieve better results in detecting visually similar app icons. Following are the main contributions of this paper.

- We show that the novel method of using combined *style* and *content* embeddings generated from pre-trained CNNs outperforms many baseline image retrieval methods for the task of detecting visually similar app icons. We also validate this method using two standard image retrieval datasets.
- Using a large dataset of over 1.2 million app icons, we show that *combined content and style embeddings* achieve 8%–12% higher *precision@k* and 14%–18% higher *recall@k*.

¹ *Temple Piggy* is currently not available in Google Play Store.

² *Temple Run* - <https://play.google.com/store/apps/details?id=com.imangi.templerun>.

- We show that adding *text embeddings* [31] as an additional modality, further increases the performance by 3%–5% and 6%–7% in terms of *precision@k* and *recall@k*.
- We identify a set of 7,246 potential counterfeits to the top-10,000 apps in Google Play and show that 2,040 of them may contain malware. We further show that out of those; 1,565 ask for at least five extra dangerous permissions and 1,407 have at least five extra third party ad libraries.

2 RELATED WORK

2.1 Mobile Malware & Greyware

While there is a plethora of work on detecting mobile malware [14, 22, 44, 52, 54] and various fraudulent activities in app markets [15, 21, 43, 47, 53], only a limited amount of work focused on the *similarity of mobile apps*. Viennot et al. [49] used the Jaccard similarity of app resources in the likes of images and layout XMLs to identify clusters of similar apps and then used the developer name and certificate to differentiate clones from rebranding. Crussell et al. [17] proposed to use features generated from the source codes to identify app clones. In contrast to above work, our work focuses on identifying visually similar apps (i.e. counterfeits) rather than the exact similarity (i.e. clones), which is a more challenging problem.

Limited amount of work focused on identifying visually similar mobile apps [7, 35, 36, 46]. For example, Sun et al. [46] proposed DroidEagle that identifies the visually similar apps based on XML layouts. While the results are interesting this method has several limitations. First, all visually similar apps may not be necessarily similar in XML layouts and it is necessary to consider the similarities in images. Second, app developers are starting to use code encryption methods, thus accessing codes and layout files may not always be possible. Third, dependency of specific aspects related to one operating system will not allow to make comparisons between heterogeneous app markets. Recently, Malisa et al. [35] studied how likely would users detect a spoofing application using a complete rendering of the application itself. In contrast to above work, the proposed work intends to use neural embeddings derived from app icons and text descriptions that will better capture visual and functional similarities.

2.2 Visual Similarity & Style Search

Several work focused on transferring style of an image to another. For example, Gatys et al. [18, 20] proposed a *neural style transfer* that is able to transfer the stylistic features of well-known artworks to target images. Other methods proposed to achieve the same objective either by updating pixels in the image iteratively or by optimising a generative model iteratively and producing the styled image through a single forward pass. A summary of style transfer algorithms can be found in the survey by Jing et al. [27].

Johnson et al. [28] proposed a feed-forward network architecture capable of real-time style transfer by solving the optimisation problem formulated by Gatys et al. [20]. Similarly, to style transfer, CNNs have been successfully used for image searching. In particular, Bell & Bala [12] proposed a Siamese CNN to learn a high-quality embedding that represent visual similarity and demonstrated the utility of these embeddings on several visual search tasks such as searching similar products. Tan et al. [48] and Matsuo & Yanai [37]

used embeddings created from CNNs to classify artistic styles. In contrast to these works, our work focuses on retrieving visually similar Android apps and we highlight the importance of style and combined (multi-modal) embeddings in this particular problem.

3 DATASET

We collected our dataset by crawling Google Play Store using a Python crawler between January and March, 2018. The crawler was seeded with the web pages of the top apps as of January, 2018 and it recursively discovered apps by following the links in the seeded pages and the subsequently discovered pages. For each app, we downloaded the metadata such as app name, app description, and number of downloads as well as the app icon in *.jpg* or *.png* format (of size 300 x 300 x 3 - height, width, and three RGB channels).

We collected information of **1,278,297 apps** during this process. For each app, we also downloaded the app executable using *Google Play Downloader* [1] by simulating a Google Pixel. We were able to download APKs for **1,023,521 apps** out of the total **1,278,297 apps** we discovered. The reasons behind this difference are the *paid apps* and the apps that did not support the used virtual device.

Labelled set: To evaluate the performance of various image similarity metrics we require a ground truth dataset that contains similar images to a given image. We used a heuristic approach to shortlist a possible set of visually similar apps and refined it by manual checking. Our heuristic is based on the fact that there are apps having multiple legitimate versions. For example, popular game app *Angry Birds* has multiple versions such as *Angry Birds Rio*, *Angry Birds Seasons*, and *Angry Birds Go* with similar app icons and descriptions.

Thus, we first identified the set of developers who has published more than two apps and one app having at least 500,000 downloads. In the set of apps from the same developer, the app with the highest number of downloads was selected as the *base app*. For each other app in the set, we calculated the *character level cosine similarity* of its *app name* to the base app name and selected the apps having over 0.8 similarity and in the same *app category* as the base app. We identified 2,689 such groups. Finally, we manually inspected each group and discarded the apps that were not visually similar.

During our later evaluations, the highest number of neighbours we retrieve is 20. Therefore, we ensured that the maximum number of apps in a group was 20 by randomly removing apps from the groups having more than 20 apps. At the end of this process we had 806 app groups having a total of 3,539 apps as our **labelled set**.

Top-10,000 popular apps: To establish a set of potential counterfeits, we used top-10,000 apps since counterfeits majorly target popular apps. We selected top-10,000 popular apps by sorting the apps by the number of downloads, number of reviews, and average rating similar to what was proposed in [42].

Other image retrieval datasets: To benchmark the performance of our proposed combined embeddings, we use two existing ground-truth datasets; UKBench [38] and Holidays [26]. Both datasets contain groups of similar images, containing images of same scenes or objects from different angles and illumination levels.

4 METHODOLOGY

The main problem we are trying to address is that “*given an app can we find potential counterfeits from a large corpus?*”. Since counterfeit

apps focus more on being visually similar to original apps, we mainly focus on finding similar app icons to a given app icon. We also focus on the similarity between text as an additional modality.

4.1 App Icon Encoding and Embeddings

We encode the original app icon image of size $300 \times 300 \times 3$ to a lower dimension for efficient search and to avoid false positives happening due to L_2 distance at large dimensions [4]. We create several low dimensional representations of the images. As baseline methods, we use state-of-the-art image *hashing methods*, *feature-based image retrieval methods*, and SSIM (Structural Similarity). From a pre-trained VGGNet, we derive *content* and *style embeddings*.

i) Hashing methods: Hashing methods we evaluate include *average* [29], *difference* [30], *perceptual* [55], and *wavelet* [5] hashing. All four methods first scale the app icon to a 32×32 grayscale image and represent it as a 1×1024 binary feature vector.

ii) Feature-based image retrieval methods: Feature-based methods extract features from an image and describe them using neighbouring pixels. Thus, such methods have two steps; *feature detection* and *feature description*. Some algorithms perform both tasks together while others perform them separately. In this paper, we use four feature matching methods; *Scale-Invariant Feature Transform (SIFT)* [34], *Speeded-Up Robust Features (SURF)* [11], *Accelerated KAZE (AKAZE)* [6], and *Learned Arrangements of Three Patch Codes (LATCH)* [32]. SIFT and SURF describe an app icon by a $f_i \times 128$ integer descriptor matrix, where f_i is the number of features detected for app icon i . AKAZE and LATCH describe the app icon by a $f_i \times 64$ binary descriptor matrix.

iii) Structural Similarity Index Matrix (SSIM): SSIM [51] compares the local pattern of pixel intensities in two images and calculate a similarity score. This method gives a high similarity score even for images with significant pixel-wise difference as it does not compare images point-by-point basis. SSIM does not represent an image by a vector/matrix. Therefore, we scale the input app icons into a 32×32 grayscale images and calculate the similarity score.

iv) Content embeddings: To extract the content representation of an icon, we used a pre-trained VGGNet [45]. We fed all 1.2M app icons to the VGGNet, and used the *content embeddings*, $C \in \mathbb{R}^{4096}$, generated at the last fully connected layer of VGGNet (fc_7 layer) that have shown good results in the past [10, 13].

v) Style embeddings: Content similarity alone is not sufficient for counterfeit detection as sometimes developers keep the visual similarity and change the content. Thus, we require an embedding that represents the style of an image.

Several work demonstrated that the Gram matrix of filter responses of CNNs can be used to represent the style of an image [19, 20]. We followed a similar approach and used the fifth convolution layer ($conv5_1$) of the VGGNet to obtain the style representation of the image, as previous work indicated that $conv5_1$ provides better performance in style similarity [37]. We passed each icon through the VGGNet, and at $conv5_1$ the icon was convolved with pre-trained filters and activated through *ReLU* function. The $conv5_1$ – layer of the VGGNet we used had 512 filters, resulting a Gram matrix of size $G^5 \in \mathbb{R}^{512 \times 512}$. We only considered the upper half of the Gram matrix as our style representation vector,

$S \in \mathbb{R}^{131,328}$ as the Gram matrix is symmetric. To further reduce the dimension of style embeddings we used the *very sparse random projection* [33] and ensured the size of style embeddings is 1×4096 . In Figure 2, we show a summary of our icon encoding process.

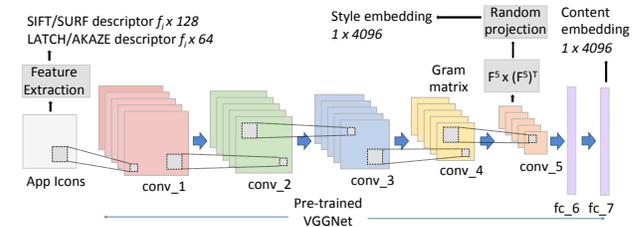


Figure 2: Image embeddings generation process

4.2 Text Embeddings

The app descriptions from Google Play Store can contain a maximum of 4000 characters, describing the apps’ functionalities and features. As such, counterfeits are likely to show some similarities to original apps’ description. To capture this similarity, we first used standard text pre-processing methods on app descriptions and trained a Paragraph Vectors Model [31] to create vectors of size 100.

4.3 Retrieving Similar Apps

During the similar app retrieval process we take an app and calculate the required embeddings and search in the encoded space for k nearest neighbours using *cosine distance*, L_2 distance, or *hamming distance* based on the applicability to the embedding under consideration. Let X_i^y be a vectored representation of an app i using the encoding scheme y and X_t^y be the corresponding representation of the target app we are comparing, we calculate the various distance metrics for different representations as summarised in Table 1.

5 RESULTS

5.1 Evaluation of Embeddings

To quantify the performance of the different embeddings, we evaluate them in four different test scenarios using multiple datasets. In each scenario, for a given query embedding, we retrieved k -nearest neighbours (k -NN) based on the distances considered in Table 1. We tested four values of k ; 5, 10, 15, and 20. The four scenarios are:

i) Holidays dataset: Holidays dataset contains 1,491 images from 500 groups. We took the encoded representation of the first image to search the entire corpus and retrieved the k -nearest neighbours.

ii) UKBench dataset: UKBench dataset contains 10,200 images from 2,550 groups. Same method as the Holidays dataset was used.

iii) Apps - Labelled set only: Labelled set contains 3,539 images from 806 groups. From each group, the *base app icon embedding* was taken as the query to retrieve k -NNs among remaining icons.

iv) Apps - Labelled set and all remaining icons and text: This dataset contains 1.2M images. The embedding of the base app icon of each group in the labelled set was taken as the query to retrieve the k -NNs from the entire dataset.

Table 1: Summary of encoding methods and distance metrics

Encoding Method	Size (n)	Distance function
Hashing methods (Hamming distance)		
Average	1,024	$\ X_i^{avg} \oplus X_t^{avg}\ _1$
Difference	1,024	$\ X_i^{diff} \oplus X_t^{diff}\ _1$
Perceptual	1,024	$\ X_i^{perc} \oplus X_t^{perc}\ _1$
Wavelet	1,024	$\ X_i^{wave} \oplus X_t^{wave}\ _1$
Feature based methods (L_2 distance)		
SIFT	$f_i^{sift} \times 128$	$\sum_{x_i \in X_i} \min_{x_t \in X_t} [\ x_i^{sift} - x_t^{sift}\ _2]$
SURF	$f_i^{surf} \times 128$	$\sum_{x_i \in X_i} \min_{x_t \in X_t} [\ x_i^{surf} - x_t^{surf}\ _2]$
Feature based methods (Hamming distance)		
AKAZE	$f_i^{akaze} \times 64$	$\sum_{x_i \in X_i} \min_{x_t \in X_t} [\ x_i^{akaze} \oplus x_t^{akaze}\ _2]$
LATCH	$f_i^{latch} \times 64$	$\sum_{x_i \in X_i} \min_{x_t \in X_t} [\ x_i^{latch} \oplus x_t^{latch}\ _2]$
Structural similarity		
SSIM	Directly returns a dissimilarity	
Neural embeddings (Cosine distance)		
Content (C_{cos})	4,096	$1 - \frac{X_i^{cont} \cdot X_t^{cont}}{\ X_i^{cont}\ _2 \ X_t^{cont}\ _2}$
Style (S_{cos})	4,096	$1 - \frac{X_i^{style} \cdot X_t^{style}}{\ X_i^{style}\ _2 \ X_t^{style}\ _2}$
Text (T_{cos})	100	$1 - \frac{X_i^{text} \cdot X_t^{text}}{\ X_i^{text}\ _2 \ X_t^{text}\ _2}$
Content+Style	8,192	$\alpha C_{cos} + \beta S_{cos}$
Content+Style+Text	8,292	$\alpha C_{cos} + \beta S_{cos} + \gamma T_{cos}$

The intuition behind above scenarios is that if a given embedding is a good representation, the k -NNs we retrieve must be from the same group as the query. Thus, for each scenario, we present **precision@k** and **recall@k**, where $k \in \{5, 10, 15, 20\}$, as the performance metrics. **Precision@k** gives the percentage of relevant images among the retrieved images. **Recall@k** is the percentage of relevant retrieved images out of the all relevant images.

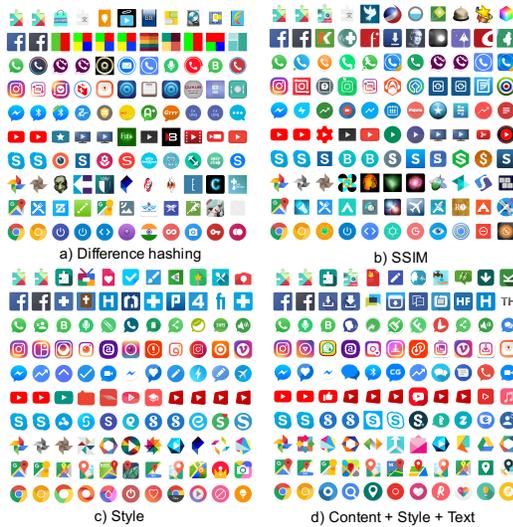


Figure 3: 10-Nearest neighbours of the top-10 popular apps

We present $precision@k$ and $recall@k$ values for all four test scenarios in Table 2 and Table 3. To choose the best β and γ values in multi-modals neural embeddings, we varied β and γ from 1 to 10

with an interval of one. We achieved the best results when $\beta = 5$ and $\gamma = 4$ and we report those results in Tables 2 and 3. The main takeaway messages from results in these two tables are:

- In all scenarios, neural embeddings outperform other methods. For example, for all four k-NN scenarios, the style embeddings have approximately 4%–14% and 11%–26% higher performance in $precision@k$ and $recall@k$ in all apps dataset.
- In UKBench and Holidays datasets, content, style, and combined embeddings increase $precision@k$ and $recall@k$ by 10%–15% and 12%–25%, respectively when retrieving five nearest neighbours. Combining style embeddings with content embeddings achieves 12% higher $precision@k$ and 14% higher $recall@k$ in all apps dataset compared to hashing and feature-based baselines when $k = 5$. Only scenario where combined content and style embeddings did not outperform all other methods is the UKBench dataset.
- It is also noticeable that adding text embedding further increases the performance by 3%-5% and 6%-7% in terms of $precision@k$ and $recall@k$, respectively, compared to the best neural embedding method when $k \in \{5, 10\}$.
- Results also show that increasing the k value increases the $recall@k$, however, significantly decreases $precision@k$. The main reason is that average number of images per groups in all four datasets is less than 5 and thus the number of false positive images in the retrieved image set increases with k.

To elaborate further on the performance of the embeddings qualitatively, in Figure 3, we present the 10-nearest neighbours we retrieved using difference hashing, SSIM, style embeddings, and content+style+text embeddings for the top-10 most popular apps in Google Play Store. Figure 3-(a) shows that hashing methods do not identify visually similar apps apart from the first 1-2 similar apps (E.g. row 9 - Google Maps). Neural embeddings based methods in Figure 3-(c) and Figure 3-(d) have identified better fits in several cases (E.g. row 1 - Google Play Services and row 9 - Google Maps). Also, Figure 3-(c) shows that style embeddings have retrieved app icons that have the same “look and feel” in terms of colour.

5.2 Retrieving Potential Counterfeits

We next use the embeddings that performed best ($Content_{cos} + \beta Style_{cos} + \gamma Text_{cos}$ where $\beta = 5$ and $\gamma = 4$) to retrieve 10-nearest neighbours for **top-10,000 apps** from the corpus of 1.2 million apps that are not from the same developer. The 10-nearest neighbour search is forced to return 10 nearest apps, irrespective of the distance. As such, there can be cases where the nearest neighbour search returns apps that are very far from the query app. Thus, we applied a distance threshold to further narrow down the results. From the retrieved 10 results for each query app, we discarded the results that are having distances greater than an empirically decided threshold. The threshold was chosen as the knee-point [40] of the cumulative distribution of all the distances with the original apps. This process returned 60,638 unique apps that are potentially counterfeits of one or more apps with in top-10,000 popular apps. Out of this 60,638 we had APK files for 49,608 apps.

		Average	Difference	Perceptual	Wavelet	SIFT	SURF	LATCH	AKAZE	SSIM	C_{cos}	S_{cos}	$C_{cos} + \beta S_{cos}$	$C_{cos} + \beta S_{cos} + \gamma T_{cos}$
Holidays	5-NN	24.56	22.68	21.60	24.48	33.00	31.12	29.16	31.00	21.88	46.36	46.72	47.92	N/A
	10-NN	13.08	11.74	10.96	12.98	17.58	16.66	15.18	15.90	11.26	25.28	25.24	25.92	N/A
	15-NN	9.00	8.08	7.36	8.92	12.15	11.55	10.43	10.91	7.76	17.47	17.25	17.89	N/A
	20-NN	6.95	6.19	5.58	6.83	9.34	8.91	7.99	8.31	5.98	13.31	13.13	13.57	N/A
UKBench	5-NN	27.29	22.44	21.63	26.37	55.27	52.97	44.82	41.77	28.46	70.22	65.01	70.06	N/A
	10-NN	15.01	11.70	10.97	14.26	28.99	27.93	23.72	21.98	15.22	36.90	33.86	36.62	N/A
	15-NN	10.51	7.95	7.38	9.99	19.82	19.12	16.22	15.04	10.55	25.03	22.95	24.79	N/A
	20-NN	8.18	6.08	5.59	7.75	15.11	14.60	12.37	11.47	8.17	18.97	17.40	18.75	N/A
Labelled	5-NN	45.14	48.41	47.62	44.44	48.92	47.67	46.63	44.22	45.34	56.43	60.42	62.23	64.76
	10-NN	23.98	28.10	27.42	25.50	26.79	27.05	26.34	25.07	25.59	33.69	35.39	36.04	38.47
	15-NN	18.59	19.92	19.45	18.08	18.86	19.00	18.45	17.54	18.06	24.05	25.25	25.57	27.19
	20-NN	14.52	15.56	15.24	14.16	14.57	14.69	14.24	13.5	14.09	18.69	19.66	19.86	21.09
All	5-NN	34.89	38.01	37.07	34.17	38.23	39.13	37.32	36.87	37.39	45.51	50.72	50.91	55.96
	10-NN	19.43	21.53	20.79	19.08	21.82	22.10	21.09	20.81	20.73	26.08	29.57	29.81	32.99
	15-NN	13.69	15.30	14.74	13.32	15.31	15.52	14.82	14.63	14.47	18.30	20.90	21.12	23.46
	20-NN	10.63	11.89	11.40	10.36	11.87	11.97	11.46	11.33	11.15	14.07	16.14	16.31	18.23

Table 2: precision@k for all test scenarios (NN* - Nearest Neighbours)

		Average	Difference	Perceptual	Wavelet	SIFT	SURF	LATCH	AKAZE	SSIM	C_{cos}	S_{cos}	$C_{cos} + \beta S_{cos}$	$C_{cos} + \beta S_{cos} + \gamma T_{cos}$
Holidays	5-NN	41.18	38.03	36.22	41.05	55.33	52.18	48.89	51.98	36.69	77.73	78.34	80.35	N/A
	10-NN	43.86	39.37	36.75	43.53	58.95	55.87	50.91	53.32	37.76	84.78	84.64	86.92	N/A
	15-NN	45.27	40.64	37.02	44.87	61.10	58.08	52.45	54.86	39.03	87.86	86.79	90.01	N/A
	20-NN	46.61	41.52	37.42	45.81	62.64	59.76	53.59	55.73	40.11	89.27	88.06	91.01	N/A
UKBench	5-NN	34.11	28.05	27.04	32.96	69.09	66.22	56.03	52.23	35.58	87.78	81.27	87.58	N/A
	10-NN	37.51	29.25	27.42	35.66	72.47	69.84	59.3	54.96	38.03	92.25	84.65	91.54	N/A
	15-NN	39.41	29.82	27.69	37.46	74.34	71.69	60.83	56.4	39.56	93.85	86.08	92.96	N/A
	20-NN	40.92	30.82	27.93	38.73	75.57	72.99	61.83	57.34	40.86	94.83	86.98	93.75	N/A
Labelled	5-NN	51.40	55.35	54.22	50.61	55.43	54.28	53.09	50.35	51.60	64.26	68.80	69.82	73.75
	10-NN	59.17	64.08	62.44	58.07	61.00	61.60	59.99	57.11	58.24	76.72	80.59	82.09	87.62
	15-NN	63.49	68.04	66.46	61.77	64.42	64.91	63.04	59.93	61.66	82.17	86.27	87.34	92.88
	20-NN	66.12	70.90	69.40	64.48	66.37	66.91	64.88	61.51	64.12	85.14	89.55	90.45	96.07
All	5-NN	39.73	42.29	42.22	38.91	43.29	44.30	42.47	41.96	42.55	51.82	57.76	57.98	63.72
	10-NN	44.25	49.03	47.36	43.46	49.42	50.04	48.01	47.36	47.19	59.40	67.34	67.90	75.13
	15-NN	46.76	52.27	50.35	45.49	52.02	52.73	50.61	49.93	49.39	62.53	71.40	72.14	80.16
	20-NN	48.43	54.14	51.94	47.19	53.75	54.23	52.16	51.57	50.75	64.09	73.52	74.29	83.05

Table 3: recall@k for all test scenarios (NN* - Nearest Neighbours)

5.3 Malware Analysis

We then checked each of the 49,608 potential counterfeits using the private API of the online malware analysis tool *VirusTotal* [2]. VirusTotal scans the APKs with over 60 commercial anti-virus tools (AV-tools) and provides a report on how many of those tools identified whether the submitted APKs contain malware. In Figure 4, we show a summary of the number of apps that were tagged as possible malware by one or more AV-tools in VirusTotal and their availability in Google Play Store as of 24-10-2018. There were 7,246 APKs that are tagged by at least one of the AV-tool.

However, a single AV-tool tagging an APK as malware may not mean that the APK contains malware. Thus, previous work used different thresholds for the number of AV-tools that must report to consider an APK as malware. Ikram et al. [23] used a conservative threshold of 5 and Arp et al. [9] used a relaxed threshold of 2. Figure 4 shows that we have 3,907 apps if the AV-tool threshold is 2 and 2,040 apps if the threshold is 5, out of which 2,067 and 1,080 apps respectively, are still there in Google Play Store. ~46% of the apps (3,358) that were tagged by at least one AV-tool are currently

not available in Google Play Store. One possible reason is that these apps were removed by Google after binary analysis. In Table 4, we show some example apps that were tagged as containing malware.

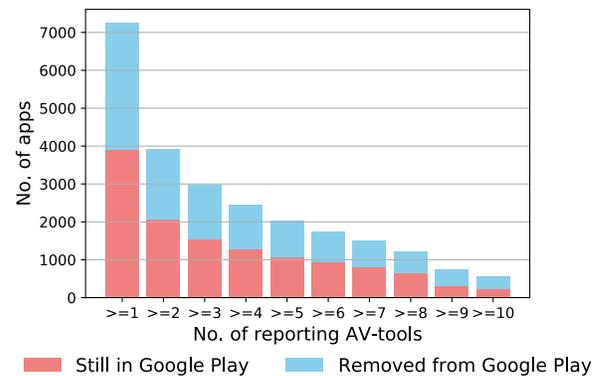


Figure 4: No. of apps against the no. of reporting AV-tools

Table 4: Example similar apps that contain malware

Original app	Similar app	AV-tools	Downloads (Original)	Downloads (Similar)
 Clean Master	 Ram Booster*	12	500 million - 1 billion	500 - 1,000
 Temple Run	 Endless Run*	12	100 million - 500 million	5,000 - 10,000
 Temple Run 2	 Temple Theft Run*	12	500 million - 1 billion	500,000 - 1 million
 Hill Climb Racing	 Offroad Racing: Mountain Climb	9	100 million - 500 million	1 million - 5 million
 Parallel Space	 Double Account*	17	50 million - 100 million	100,000 - 500,000

*The app is currently not available in Google Play Store

5.4 Permission Requests

Another motivation behind counterfeiting can be collecting personal data. We considered the 26 dangerous Android permissions [3] and to identify the potential counterfeits that ask for more permissions than the original app, we define a metric, *permissions difference*, which is the difference between the number of dangerous permissions requested by the potential counterfeit but not the original app and number of dangerous permissions requested by the original app but not by the potential counterfeit app. If the *permissions difference* is a positive value that means the counterfeit asks for more dangerous permissions than the original app and vice versa if it is negative. For the 49,608 potential counterfeits we had the APK files, we calculated the *permission difference*.

The cumulative sum of number of apps against the *permission difference* is shown in Figure 5a. The majority of the potential counterfeits did not ask for more dangerous permissions than the original app. However, there are 17,230 potential counterfeits that are asking at least one dangerous permission than the original app (13,857 unique apps), and 1,866 potential counterfeits (1,565 unique apps) asking at least five additional dangerous permissions. In Figure 5b we show Google Play Store availability of the 17,230 apps with positive *permissions difference* as of 24-10-2018. Figure shows approximately 37% of the potential counterfeits with a permission difference of five is currently not available in the Google Play Store.

5.5 Advertisement Libraries

Another motivation behind developing counterfeits can be monetisation using advertisements. To quantify this, we defined; *ad library difference* using the list of 124 mobile advertising and analytics libraries provided in [41]. *Ad library difference* is the difference between the number of advertisement libraries embedded in the potential counterfeit but not in the original app and

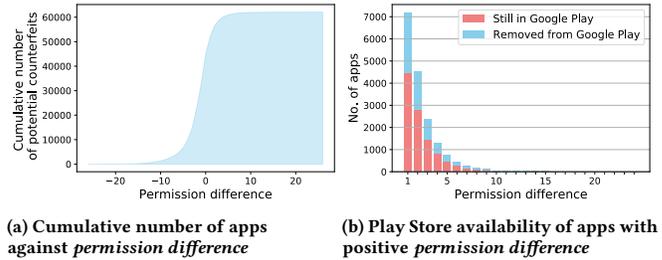


Figure 5: Counterfeits requesting extra permissions

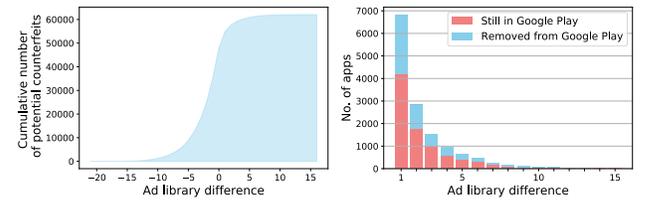


Figure 6: Counterfeits with additional ad libraries

number of advertisement libraries embedded in the original app but not in the potential counterfeit app. We show the cumulative number of counterfeits over the range of *ad library difference* in Figure 6a. According to the figure, 13,997 apps (11,281 unique apps) have a positive ad library difference and out of that 1,841 (1,407 unique apps) have an ad library difference greater than or equal to five. Figure 6b shows the Google Play store availability of apps with a positive ad library difference. Approximately 33% of the apps we identified are currently not available in the Google Play Store.

6 CONCLUSION

We proposed an icon encoding method that allows to efficiently search potential counterfeits to a given app, using neural embeddings generated by CNNs. Specifically, for app counterfeit detection problem, we showed that content and style neural embeddings generated from a pre-trained VGGNet significantly outperforms hashing and feature-based image retrieval methods.

We used our multi-modal embedding method to retrieve potential counterfeits for the top-10,000 apps in Google Play and investigated the possible inclusion of malware, permission usage, and embedded third party ad libraries. We found that 2,040 potential counterfeits we retrieved were marked by at least five commercial antivirus tools as malware, 1,565 asked for at least five additional dangerous permissions, and 1,407 had at least five additional embedded third party ad libraries. Finally, we showed that as of now (6-10 months since we discovered the apps), 27%–46% of the potential counterfeits we identified are not available in Google Play Store, potentially removed due to customer complaints.

ACKNOWLEDGEMENT

This project is partially funded by the Google Faculty Rewards 2017, NSW Cyber Security Network’s Pilot Grant Program 2018, and the Next Generation Technologies Program. Authors would

like to thank VirusTotal for kindly providing access to the private API, which was used for the malware analysis in this paper.

REFERENCES

- [1] 2018. <https://github.com/matlink/gplaycl>.
- [2] 2018. <https://www.virustotal.com>.
- [3] 2018. <https://developer.android.com/guide/topics/permissions>.
- [4] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. 2001. On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT*. Springer.
- [5] Fawad Ahmed and M. Y. Sial. 2006. A Secure and Robust Wavelet-Based Hashing Scheme for Image Authentication. In *Advances in Multimedia Modeling*, 51–62.
- [6] Pablo Fernández Alcantarilla, Jesús Nuevo, and Adrien Bartoli. 2013. Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces. In *BMVC*, 1–9.
- [7] Benjamin Andow, Adwait Nadkarni, Blake Bassett, William Enck, and Tao Xie. 2016. A study of grayware on Google Play. In *Security and Privacy Workshops (SPW), 2016 IEEE*. IEEE.
- [8] Ionut Arghire. 2017. Fake Netflix App Takes Control of Android Devices. <http://www.securityweek.com/fake-netflix-app-takes-control-android-devices>.
- [9] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *NDSS*.
- [10] Artem Babenko, Anton Slesarev, Alexander Chigorin, and Victor S. Lempitsky. 2014. Neural Codes for Image Retrieval. *CoRR* abs/1404.1777 (2014). arXiv:1404.1777
- [11] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. SURF: Speeded Up Robust Features. In *Computer Vision-ECCV*. Springer Berlin Heidelberg, 404–417.
- [12] Sean Bell and Kavita Bala. 2015. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)* (2015).
- [13] Sean Bell and Kavita Bala. 2015. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)* (2015).
- [14] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. Crowdroid: Behavior-based malware detection system for android. In *Proc. of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 15–26.
- [15] Rishi Chanday and Haijie Gu. 2012. Identifying spam in the iOS app store. In *Proc. of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*. ACM, 56–59.
- [16] Sam Costello. 2018. How Many Apps Are in the App Store? <https://www.lifewire.com/how-many-apps-in-app-store-2000252>. Accessed: 2018-04-12.
- [17] Jonathan Crussell, Clint Gibler, and Hao Chen. 2013. Andarwin: Scalable detection of semantically similar Android applications. In *European Symposium on Research in Computer Security*. Springer, 182–199.
- [18] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015).
- [19] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. Texture synthesis and the controlled generation of natural stimuli using convolutional neural networks. *CoRR* abs/1505.07376 (2015). arXiv:1505.07376
- [20] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. 2016. Image style transfer using Convolutional Neural Networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*.
- [21] Clint Gibler, Ryan Stevens, Jonathan Crussell, Hao Chen, Hui Zang, and Heesook Choi. 2013. Adrob: Examining the landscape and impact of Android application plagiarism. In *Proc. of the 11th MobiSys*. ACM.
- [22] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. Riskranker: Scalable and accurate zero-day Android malware detection. In *Proc. of the 10th international conference on Mobile systems, applications, and services*. ACM, 281–294.
- [23] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. 2016. An Analysis of the Privacy and Security Risks of Android VPN Permission-enabled Apps. In *Proc. of the 2016 ACM on Internet Measurement Conference*.
- [24] Statista Inc. 2018. Number of available applications in the Google Play Store from December 2009 to December 2017. <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>.
- [25] Chris Jager. 2018. Scam Alert: Fake CBA And ANZ Bank Apps Discovered On Google Play Store. <https://www.lifehacker.com.au/2018/09/scam-alert-fake-cba-and-anz-banking-apps-found-on-google-play-store/>. Accessed: 2018-10-15.
- [26] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2008. Hamming embedding and weak geometric consistency for large scale image search. In *European conference on computer vision*. Springer, 304–317.
- [27] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, and Mingli Song. 2017. Neural Style Transfer: A Review. *arXiv preprint arXiv:1705.04058* (2017).
- [28] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*. Springer, 694–711.
- [29] Neal Krawetz. 2013. <http://www.hackerfactor.com/blog/?/archives/432-Looks-Like-It.html>.
- [30] Neal Krawetz. 2013. <http://www.hackerfactor.com/blog/?/archives/529-Kind-of-Like-That.html>.
- [31] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proc. of the 31st ICML*. 1188–1196.
- [32] G. Levi and T. Hassner. 2016. LATCH: Learned arrangements of three patch codes. In *IEEE Winter Conference on Applications of Computer Vision*. 1–9.
- [33] Ping Li, Trevor J Hastie, and Kenneth W Church. 2006. Very sparse random projections. In *Proc. of the 12th ACM SIGKDD*. ACM, 287–296.
- [34] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60, 2 (2004).
- [35] Luka Malisa, Kari Kostiaainen, and Srdjan Capkun. 2017. Detecting Mobile Application Spoofing Attacks by Leveraging User Visual Similarity Perception. In *Proc. of the Seventh ACM on Conference on Data and Application Security and Privacy (CODASPY '17)*. ACM, New York, NY, USA, 289–300. <https://doi.org/10.1145/3029806.3029819>
- [36] Luka Malisa, Kari Kostiaainen, Michael Och, and Srdjan Capkun. 2016. Mobile application impersonation detection using dynamic user interface extraction. In *European Symposium on Research in Computer Security*. Springer, 217–237.
- [37] Shin Matsuo and Keiji Yanai. 2016. CNN-based style vector for style image retrieval. In *Proc. of the 2016 ACM ICMR*. ACM, 309–312.
- [38] David Nister and Henrik Stewenius. 2006. Scalable Recognition with a Vocabulary Tree. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.
- [39] Sarah Perez. 2013. Developer Spams Google Play With Ripoffs Of Well-Known Apps Again. <http://techcrunch.com>.
- [40] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. 2011. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*. IEEE.
- [41] Suranga Seneviratne, Harini Kolumunna, and Aruna Seneviratne. 2015. A measurement study of tracking in paid mobile applications. In *Proc. of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, 7.
- [42] Suranga Seneviratne, Aruna Seneviratne, Mohamed Ali Kaafar, Anirban Mahanti, and Prasant Mohapatra. 2015. Early detection of spam mobile apps. In *Proc. of the 24th International Conference on World Wide Web*.
- [43] Suranga Seneviratne, Aruna Seneviratne, Mohamed Ali Kaafar, Anirban Mahanti, and Prasant Mohapatra. 2017. Spam Mobile Apps: Characteristics, Detection, and in the Wild Analysis. In *To Appear in Proc. of Transactions on the Web (TWEB)*. ACM.
- [44] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. 2012. Andromaly: A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems* 38, 1 (2012), 161–190.
- [45] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556
- [46] Mingshen Sun, Mengmeng Li, and John Lui. 2015. DroidEagle: Seamless detection of visually similar Android apps. In *Proc. of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM.
- [47] Didi Surian, Suranga Seneviratne, Aruna Seneviratne, and Sanjay Chawla. 2017. App Misclassification Detection: A Case Study on Google Play. *IEEE TKDE* 29, 8 (2017).
- [48] Wei Ren Tan, Chee Seng Chan, Hernán E Aguirre, and Kiyoshi Tanaka. 2016. Ceci n'est pas une pipe: A deep convolutional network for fine-art paintings classification. In *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE.
- [49] Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A measurement study of Google Play. In *ACM SIGMETRICS Performance Evaluation Review*. ACM.
- [50] Kyle Wagner. 2012. Fake Angry Birds Space Android App Is Full Of Malware. <https://www.gizmodo.com.au/2012/04/psa-fake-angry-birds-space-android-app-is-full-of-malware/>.
- [51] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.
- [52] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*. IEEE, 62–69.
- [53] Zhen Xie and Sencun Zhu. 2015. AppWatcher: Unveiling the underground market of trading mobile app reviews. In *Proc. of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM.
- [54] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. DroidSec: Deep learning in Android malware detection. In *ACM SIGCOMM Computer Communication Review*.
- [55] H. Zhang, M. Schmucker, and X. Niu. 2007. The Design and Application of PHABS: A Novel Benchmark Platform for Perceptual Hashing Algorithms. In *IEEE International Conference on Multimedia and Expo*. 887–890.
- [56] Yajin Zhou and Xuxian Jiang. 2012. Dissecting Android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE.