PERCIVAL: Making In-Browser Perceptual Ad Blocking Practical with Deep Learning

Zain ul Abi Din[*] *UC Davis*
Panagiotis Tigas[†] *University of Oxford*
Samuel T. King *UC Davis*
Benjamin Livshits *Brave Software* *Imperial College London*

Abstract

Online advertising has been a long-standing concern for user privacy and overall web experience. Several techniques have been proposed to block ads, mostly based on filter-lists and manually-written rules. While a typical ad blocker relies on manually-curated block lists, these inevitably get out-of-date, thus compromising the ultimate utility of this ad blocking approach.

In this paper we present PERCIVAL, a browser-embedded, lightweight, deep learning-powered ad blocker. PERCIVAL embeds itself within the browser's image rendering pipeline, which makes it possible to intercept every image obtained during page execution and to perform blocking based on applying machine learning for image classification to flag potential ads.

Our implementation inside both Chromium and Brave browsers shows only a minor rendering performance overhead of 4.55%, demonstrating the feasibility of deploying traditionally heavy models (i.e. deep neural networks) inside the critical path of the rendering engine of a browser. We show that our image-based ad blocker can replicate EasyList rules with an accuracy of 96.76%. To show the versatility of the PERCIVAL's approach we present case studies that demonstrate that PERCIVAL 1) does surprisingly well on ads in languages other than English; 2) PERCIVAL also performs well on blocking first-party Facebook ads, which have presented issues for other ad blockers. PERCIVAL proves that image-based perceptual ad blocking is an attractive complement to today's dominant approach of block lists.

1 Introduction

Online advertising is a dominant force on the web. However, many studies have shown that ads impose significant privacy and performance costs to users, and carry the potential to be a malware delivery vector [3, 28]. Users find ads intrusive [54] and these disrupt the browsing experience [6]

Several ad blocking techniques have been proposed in the literature [32]; the majority of these are based on "handcrafted" filter lists such as EasyList [66], which contain rules matching ad-carrying URLs (often JavaScript, images, and HTML) and ad-related DOM elements. These block lists are deployed in the most widely-used ad blockers, such as uBlock Origin and Adblock Plus.

Perceptual ad-blocking relies on "visual cues" frequently associated with ads like the AdChoices^{TM} logo or a sponsored content link. The concept of perceptual ad blocking is predicated on the idea that, ultimately, the intended audience of an ad is the end-user, so ads should be easily recognizable by humans [60]. To effectively evade a perceptual ad blocker, the ad content would need to be distorted significantly [30], which may both degrade the end-user experience and may also fall in violation of legal requirements [11].

Storey et al. [61] built the first perceptual ad-blocker that uses traditional computer vision techniques to detect ad-identifiers. Recently, Adblock Plus developers built filters into their ad-blocker [14] to match images against a fixed template to detect ad labels. Due to variations in ad-disclosures, AdChoices logo and other ad-identifiers, it is unlikely that traditional computer vision techniques are sufficient and generalizable to the unseen. A natural extension to these techniques is Deep Learning. Deep learning has been shown to outperform traditional vision techniques on tasks like image classification [43] and object detection [64]

In this paper, we present a native, deep learning-powered *perceptual ad blocker* called PERCIVAL, which blocks ads based on their visual appearance. We demonstrate that our model generalizes well across datasets collected using different crawling techniques at varying times. In addition to high detection accuracy, our focus is on producing small and fast models, which can be deployed online within an actual browser to do real-time ad detection and blocking. PERCIVAL can be run *in addition* to an existing ad blocker, as a last-step measure to block whatever slips through its filters. However, PERCIVAL may also be deployed *outside* the browser, for example, as part of a crawler, whose job

[0] Author was employed by Brave software when this work took place.

is to construct comprehensive block lists to supplement EasyList. We conclude that PERCIVAL provides an attractive complement to the current method of blocking ads using crowd-sources filter lists.

1.1 Contributions

This paper makes the following contributions

- **Perceptual ad blocking in Chromium-based browsers.** This paper presents PERCIVAL, an approach to last-step perceptual ad blocking designed to supplement block list-based blocking techniques. PERCIVAL analyzes images obtained *during* page rendering, using deep neural networks. We demonstrate two deployment scenarios in this paper, both implemented and tested in two Chromium-based browsers, Chrome and Brave: one blocking ads as we render the page, with an extra performance overhead. The alternative low-latency approach we propose is classifying images *asynchronously*, which allows for memoization of the results, thus speeding up the classification process. We make the source code and pre-trained models available for other researchers at https://github.com/dxaen/percival.
- **Lightweight and accurate deep learning models.** We show that ad blocking can be done effectively using highly-optimized deep neural network-based models for image processing. Previous studies have indicated that models over 5 MB in size become hard to deploy on mobile devices; because of our focus on affordable low-latency detection, we create a compressed in-browser model that is smaller by factor of 74, compared to other models of this kind [20] without a significant loss in accuracy.
- **Accuracy and performance overhead measurements.** We show that our perceptual ad-blocking model can replicate EasyList rules with the accuracy of 96.76%, making PERCIVAL into a viable and complementary ad-blocking layer. Our implementation within Chromium shows an average overhead of 178.23ms for page rendering. This overhead, although non-negligible, it shows the feasibility of deploying traditionally heavy models (i.e. deep neural networks) inside the critical path of the rendering engine of the browser.
- **First-party ad blocking.** While the focus on traditional ad blocking is primarily on third-party ad blocking, PERCIVAL is capable of blocking first-party ads, such as those found on Facebook. Specifically, our experiments show that PERCIVAL blocks ads on Facebook (often referred to as "sponsored content") with 70% recall and precision of 78.4%.
- **Language-agnostic blocking.** We demonstrate that our model in PERCIVAL is capable of blocking images that are in languages we did not train our model on. We evaluate our trained model on Arabic, Chinese, Korean, French and Spanish datasets. Our model achieves an accuracy of 81.3% on Arabic, 95.1% on Spanish, and 93.9% on French datasets, with moderately high precision and recall. This is an important result as it illustrates the out-of-the box benefit of using PERCIVAL for languages that have much lower coverage of EasyList rules, compared to the English ones.

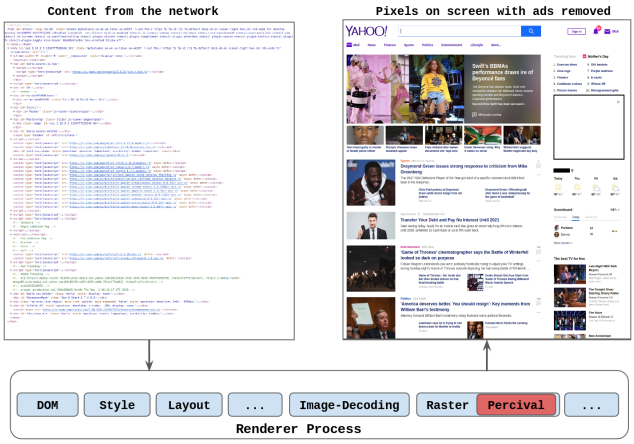


Figure 1: Overall architecture of PERCIVAL. PERCIVAL is positioned in the renderer process-which is responsible for creating rasterized pixels from HTML, CSS, JavaScript. As the renderer process creates the DOM and decodes and rasterizes all image frames, these are first passed through PERCIVAL. PERCIVAL blocks the frames that are classified as ads. The corresponding output with ads removed is shown above(right).

1.2 Paper Organization

The rest of the paper is structured as follows. Section 2 provides an overview of PERCIVAL's architecture. In Section 3 we describe our methodology for developing and compressing our perceptual ad-blocking model. We discuss the details of implementing the model inside the Blink rendering engine in Section 4 and in Section 5 evaluate the accuracy of the PERCIVAL and also the rendering performance of deploying inside the browser. In Section 6 we discuss the implications of our results and talk about deployment of PERCIVAL. In Section 7 we review related work and, finally, in Section 8 we conclude.

2 PERCIVAL Overview

This paper presents PERCIVAL, a new system for blocking ads. Our primary goal is to build a system that blocks ad images that might be allowed by current detection techniques, while remaining small and efficient enough to run in a mobile browser. Figure 1 shows how PERCIVAL blocks rendering of ads. First, PERCIVAL runs in the browser image rendering

pipeline. By running in the image rendering pipeline, we can ensure that we inspect all images before the browser shows them to the user. Second, PERCIVAL uses a deep convolutional neural network (CNN) for detecting ad images. Using CNNs enables PERCIVAL to detect a wide range of ad images, even if they are in a language that PERCIVAL was not trained on.

This section discusses PERCIVAL's architecture overview, alternative possible implementations and detection model. Section 3 discusses the detailed design and implementation for our browser modifications and our detection model.

2.1 PERCIVAL's Architecture Overview

PERCIVAL 's detection module runs in the browser's image decoding pipeline, after the browser has decoded the image into pixels, but before it displays these pixels to the user. Running PERCIVAL after the browser has decoded an image takes advantage of the browser's mature, efficient, and extensive image decoding logic, while still running at a choke point before the browser actually displays the decoded pixels. Simply put, if a user sees an image, it goes through this pipeline first.

More concretely, as shown in Figure 1 PERCIVAL runs in the renderer process of the browser engine. The renderer process on receiving the content of the web page proceeds to create the intermediate data structures to represent the web page. These intermediate representations include the DOM-which encodes the hierarchical structure of the web page, the layout-tree, which includes the layout information of all the elements of the web page, and the display-list, which includes commands to draw the elements on the screen. If an element has an image contained within it, it needs to go through the *Image Decoding Step* before it can be rasterized. We run PERCIVAL after the *Image Decoding Step* during the *raster* phase which helps run PERCIVAL in parallel for multiple images at a time. Images that are classified as ads are blocked from rendering. The web page with ads removed is shown in Figure 1 (right). We present detailed design and implementation in Section 3

2.2 Alternative Possible Implementations and Advantages of PERCIVAL

One alternative to running PERCIVAL directly in the browser could have been to run PERCIVAL in the browser's JavaScript layer via an extension. However, to run PERCIVAL's logic in JavaScript would require scanning the DOM to find image elements, waiting for them to finish loading, and then screen-shotting the pixels to run the detection model. The advantage of a JavaScript-based system is that it works within current browser extensibility mechanisms, but recent work has shown how attackers can evade this style of detection [65].

Ad blockers that inspect web pages based on the DOM such as Ad Highlighter [61] are prone to DOM obfuscation attacks. They assume that the elements of the DOM strictly correspond to their visual representation. For instance, an ad-blocker that retrieves all `img` tags and classifies the content contained in these elements does not consider the case, where a rendered image is a result of several CSS or JavaScript transformations and not the source contained in the tag.

These ad blockers are also prone to resource exhaustion attacks where the publisher injects a lot of dummy elements in the DOM to overwhelm the ad blocker.

Additionally, a native implementation is much faster than a browser extension implementation. A browser extension only has access to the browser functionality exposed in JavaScript layer. We also have access to the unmodified image buffers with a native implementation.

2.3 Detection Model

PERCIVAL runs a detection model on every image loaded in the document's main frame, a sub-document such as an `iframe`, as well as images loaded in JavaScript to determine if the image is an ad.

Although running directly within the browser provides PERCIVAL with more control over the image rendering process, it introduces a challenge: how to run the model efficiently in a browser. Our goal is to run PERCIVAL in browsers that run on laptops or even mobile phones. This requires that the model be small to be practical [56]. This design also requires that the model model run directly in the image rendering pipeline, so overhead must be low. Any overhead adds latency to rendering for all images it inspects.

In PERCIVAL, we use the SqueezeNet CNN as the starting point for our detection model. The SqueezeNet inventors designed it to be small (around 5 MB) and run efficiently on power-constrained mobile devices, yet still perform well on image classification tasks. We modify the basic SqueezeNet network to be optimized for ad blocking by removing less important layers. This results in a model size that is less than 2 MB and detects ad images in 11 ms.

A second challenge in using small CNNs is how to provide enough training data. In general, smaller CNNs can have suitable performance, but require more training data. Gathering ad images is non-trivial for a number of reasons, the number of ad and non-ad images on most web pages is largely skewed in favor of non-ad images. Most ads are programatically inserted into the document through `iframes` or JavaScript and so simple crawling methods that work only on the initial HTML of the document will miss most of the ad images.

To crawl ad images, other researchers [20, 65] propose screen-shotting iframes or JavaScript elements. This data collection method leads to problems with synchronizing the timing of the screenshot and when the element loads. Many screen-shots end up with whites-space instead of the image content. Also, this method only makes sense if the input to

the classifier is the rendered content of the web page.

To address these concerns and to provide ample training data, we design and implement a custom crawler in Blink that handles dynamically-updated data and eliminates the race condition between the browser displaying the content and the screenshot we use to capture the image data. Our custom-crawler downloads and labels ad and non-ad images directly from the rendering pipeline.

3 Design and Implementation of PERCIVAL

This section covers the design and implementation of the browser portion of PERCIVAL. We first cover the high-level design principles that guide our design, then we discuss rendering and image handling in Blink, the rendering engine of Google Chrome and the Brave browser. Finally, we describe our end-to-end implementation within Blink.

3.1 Design Goals

We have two main goals in our design of PERCIVAL:

Run PERCIVAL at a choke point: Advertisers can serve ad images in different formats, such as JPG, PNG, or GIF. Depending on the format of the image, an encoded frame can traverse different paths in the rendering pipeline. Also, a wide range of web constructs can cause the browser to load images, including HTML image tags, JavaScript image objects, HTML Canvas elements, or CSS background attributes. Our goal is to find a single point in the browser to run PERCIVAL, such that it inspects all images, operates on pixels instead of encoded images, but does so before the user sees the pixels on the screen, enabling PERCIVAL to block ad image cleanly. **Note:** If individual pixels are drawn programmatically on canvas, PERCIVAL will not block it from rendering.

In Blink, the raster task within the rendering pipeline enables PERCIVAL to inspect, and potentially block, all images. Regardless of the image format or how the browser loads it, the raster task decodes the given image into raw pixels, which it then passes to the GPU to display the content on the screen. We run PERCIVAL at this precise point to abstract different image formats and loading techniques, while still retaining the opportunity to block an image before the user sees it.

Run multiple instances of PERCIVAL in parallel: Running PERCIVAL in parallel is a natural design goal because PERCIVAL makes all image classification decisions independently based solely on the pixels of each individual image. When designing PERCIVAL, we look for opportunities to exploit this natural parallelism to minimize the latency added due to the addition of our ad blocking model.

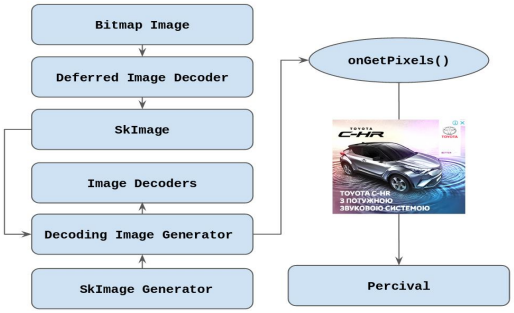


Figure 2: PERCIVAL in the image decoding pipeline. SkImage Generator allocates a bitmap and calls the `onGetPixels()` of `DecodingImageGenerator` to populate the bitmap. This bitmap is then passed to the network for classification and cleared if it contains an ad.

3.2 Rendering and PERCIVAL: Overview

We integrate PERCIVAL into Blink, the rendering engine for Google Chrome and Brave. From a high level, Blink's primary function is to turn a web page into the appropriate GPU calls [5] to show the user the rendered content.

A web page can be thought of as a collection of HTML, CSS, and JavaScript code, which the browser fetches from the network. The rendering engine parses this code to build the DOM and layout tree, and to issue OpenGL calls via `Skia`, Google's graphics library [22].

The layout tree contains the locations of the regions the DOM elements will occupy on the screen. This information together with the DOM element is encoded as a `display item`. The browser follows this parsing process by the rasterization process, which takes the display items and turns them into bitmaps. Rasterization issues OpenGL draw calls via the Skia library to draw bitmaps. If the display list items have images in them (a common occurrence), the browser must decode these images before drawing them via Skia.

PERCIVAL intercepts the rendering process at this precise point, after the `Image Decode Task` and during the `Raster Task`. As the renderer process creates the DOM and decodes and rasterizes all image frames, these are first passed through PERCIVAL. PERCIVAL blocks the frames that are classified as ads.

3.3 End-to-End Implementation in Blink

In our Blink instrumentation, we deal with Skia and Blink classes. Most of the code (forward pass of the CNN) resides at the same directory level as Blink.

Skia uses a set of image decoding operations to turn `SkImages`, which is the internal class type within Skia that encapsulates images, into bitmaps. PERCIVAL reads these bitmaps and classifies their content accordingly. If PERCIVAL classifies the bitmap as an ad, we block it by removing its content. Otherwise, PERCIVAL lets it pass through to the next layers of the rendering process. In case the content is cleared,

we have several options on how to fill up the surrounding white-space. We can either collapse it by propagating the information upwards or display a predefined image (user's spirit animal) in place of the ad.

Figure 2 shows an overview of our Blink integration. Blink class `BitmapImage` creates an instance of `DeferredImageDecoder` which in turn instantiates a `SkImage` object for each encoded image. SkImage creates an instance of `DecodingImageGenerator` (blink class) which will in turn decode the image using the relevant image decoder from Blink. Note that the image hasn't been decoded yet since chromium practices deferred image decoding.

Finally, `SkImageGenerator` allocates bitmaps corresponding to the encoded `SkImage`, and calls `onGetPixels()` of `DecodingImageGenerator` to decode the image data using the proper image decoder. This method populates the buffer (pixels) that contain decoded pixels, which we pass to PERCIVAL along with the image height, width, channels information (`SKImageInfo`) and other image metadata. PERCIVAL reads the image, scales it to $224 \times 224 \times 4$ (default input size expected by SqueezeNet), creates a tensor, and passes it through the CNN. If PERCIVAL determines that the buffer contains an ad, it clears the buffer, effectively blocking the image frame.

Rasterization, image decoding, and the rest of the processing happen on a raster thread. Blink rasters on a per tile basis and each tile is like a resource that can be used by the GPU. In a typical scenario there are multiple raster threads each rasterizing different raster tasks in parallel. PERCIVAL runs in each of these worker threads after image decoding and during rasterization, which runs the model in parallel.

As opposed to Sentinel [58] and Ad Highlighter [29] the input to PERCIVAL is not the rendered version of web content; PERCIVAL takes in the Image pixels directly from the image decoding pipeline. This is important since with PERCIVAL we have access to unmodified image buffers and it helps prevent attacks where publishers modify content of the webpage (including iframes) with overlaid masks (using CSS techniques) meant to fool the the ad blocker classifier.

4 Deep Learning Pipeline

This section covers the design of PERCIVAL's deep neural network, as well as the training workflow for developing our ad-blocking system. We first introduce the building blocks of a traditional convolution neural network, we then describe the network employed by PERCIVAL and the training process. Finally, we describe our data acquisition and labelling techniques.

4.1 Background and Terminology

Convolution neural networks (CNN) [44] are a variant of neural networks commonly used for visual recognition. The input is usually an image and the network outputs a set of probabilities. The intermediate values generated in the process are called *feature maps* and can be thought of as multiple images stacked together.

Due to the higher dimensionality of visual data, CNNs share learned parameters across the network. The input images are also down-sampled as these pass through the network. This is accomplished by convolving the input image with small shared filters and then down-sampling the output to the desired dimension. Accordingly, convolution and down-sampling constitute the two basic operations in a CNN. Each convolution filter detects a feature over some region of the input image, while also preserving the spatial locality of the image.

The down-sampling or pooling layer either computes an average or maximum value over some neighborhood of the input. The value computed is representative of the neighborhood and can be used in its place. This helps to reduce the dimensionality of the input feature maps, while also making the network robust to small variations. Additionally, as with traditional neural networks CNNs also feature a SoftMax layer; this layer turns the output feature maps into probabilities that sum to 1. Normalizing the output feature maps helps reason about the likelihood of one class over another.

4.2 PERCIVAL's CNN Architecture

We cast ad detection as a traditional image classification problem, where we feed images into our model and it classifies them as either being (1) an ad, or (2) not an ad. CNNs are the current standard in the computer vision community for classifying images. As such, we tried several standard CNNs on our dataset, including Inception-V4 [62], Inception [63], and ResNet-52 [33]. With minor modifications, all of these standard networks classify ads accurately (97–99% accuracy) on our datasets. However, the model size and the classification time of these systems was prohibitive for our purposes.

Because of the prohibitive size and speed of standard image classifiers, in PERCIVAL we use a small network, SqueezeNet [39], as the starting point for our in-browser model. The SqueezeNet authors show that SqueezeNet achieves comparable accuracy to much larger CNNs, like AlexNet, and boasts a final model size of 4.8 MB. This reduced model size makes SqueezeNet a compelling starting point for the PERCIVAL model.

SqueezeNet consists of multiple *fire modules*. A *fire module* consists of a "squeeze" layer, which is a convolution layer with 1×1 filters that reduces the number of channels by a factor of $\frac{1}{4}$ to $\frac{1}{8}$. This is followed by two "expand" convolution layers with filter sizes of 1×1 and 3×3, respectively, that increase the number of channels by a factor of $\frac{1}{4}$ each. Overall, *fire modules* reduce the number of input channels to larger convolution filters in the pipeline. Traditionally, larger

filters are needed to capture the spatial information in an image. However, these filters also increase the number of operations and classification time (forward passes in neural networks terminology).

A visual summary of network structure is shown in Figure 3. Our modified network consists of a convolution layer, followed by 6 *fire modules* and a final convolution layer, a global average pooling layer and a SoftMax layer. As opposed to the original SqueezeNet, we down-sample the feature maps at regular intervals in the network. This helps reduce the classification time per image. We also perform max-pooling after the first convolution layer and after every two fire modules.

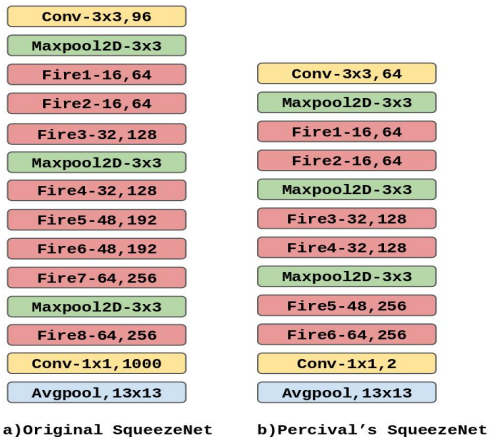
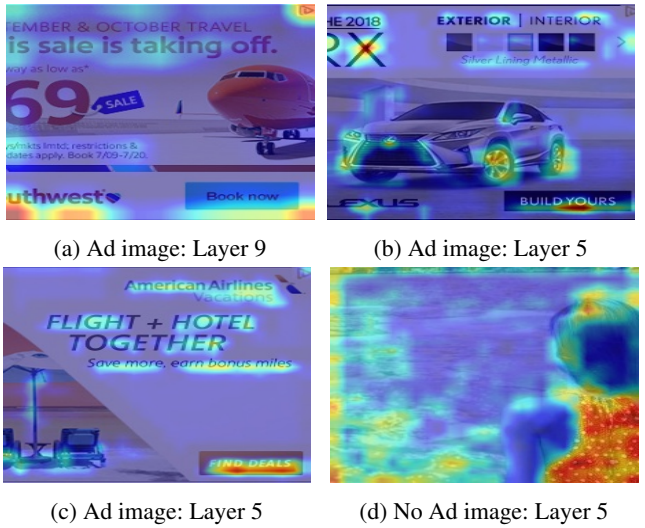


Figure 3: Original SqueezeNet (left) and PERCIVAL's fork of SqueezeNet (right). For `Conv`, `Maxpool2D`, and `Avgpool` blocks $a \times b$ represents the dimensions of the filters used. For fire blocks *a, b* represents the number of intermediate and output channels. We remove extraneous blocks as well as downsample the feature maps at regular intervals to reduce the classification time per image.



(a) Ad image: Layer 9 (b) Ad image: Layer 5

(c) Ad image: Layer 5 (d) No Ad image: Layer 5

Figure 4: Salience map of the network on a sample ad and no-ad images. Each image corresponds to the output of Grad-CAM [57] for the layer in question.

4.3 Training and Fine-Tuning

Before training the network in PERCIVAL, we initialized the blocks Convolution 1, Fire1, Fire2, Fire3, and Fire4, using the weights from a SqueezeNet model pre-trained with ImageNet [43]. The goal of using the weights from a previously-trained model is to reuse the feature extraction mechanism (also known as representation learning [26]) that was trained on the original network with a much bigger dataset. This way we can continue training on examples that are only relevant to the task we are trying to solve, therefore avoiding the need for a very large dataset.

To determine the hyper parameters for training PERCIVAL we started with the recommendations from Bengio et al. [25]. This was followed by baby-sitting the training process to manually determine the correct configuration of training hyper parameters. Ultimately, we trained PERCIVAL with stochastic gradient descent, momentum ($\beta = 0.9$), learning rate 0.001, and batch size of 24. We also used step learning rate decay and decayed the learning rate by a multiplicative factor 0.1 after every 30 epochs.

4.4 Data Acquisition

We use two systems to collect training image data. First, we use a traditional crawler and traditional ad-blocking rules (EasyList [7]) to identify ad images. Second, we use our browser instrumentation from PERCIVAL to collect images, improving on some of the issues we encountered with our traditional crawler.

4.4.1 Crawling with EasyList

We use a traditional crawler matched with a traditional rule-based ad blocker to identify ad content for our first dataset. In particular, to identify ad elements which could be iframes or complex JavaScript constructs, we use EasyList, which is a set of rules that identify ads based on the URL of the elements, location within the page, origin, class or id tag, and other hand-crafted characteristics known to indicate the presence of ad content.

We built a crawler using Selenium [19] for browser automation. We then use the crawler to visit Alexa top-1,000 web sites, waiting for 5 seconds on each page, and then randomly selecting 3 links and visiting them, while waiting on each page for a period of 5 seconds as before. For every visit, the crawler applies every EasyList network, CSS and exception rule.

For every element that matches an EasyList rule, our crawler takes a screenshot of the component, cropped tightly to the coordinates reported by Chromium, and then stores

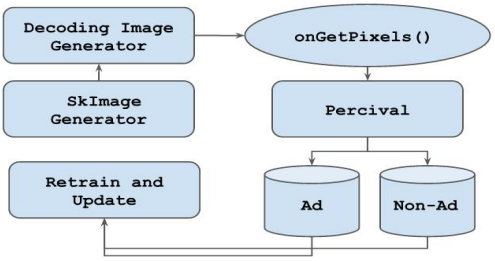


Figure 5: Crawling, labelling and re-training with PERCIVAL. Every decoded image frame is passed through PERCIVAL and PERCIVAL downloads the image frame into the appropriate bucket.

it as an ad sample. We capture non-ad samples by taking screenshots of the elements that do *not* match any of the EasyList rules. Using this approach we, extract 22,670 images out of which 13,741 are labelled as ads, and 8,929 as non-ads. This automatic process was followed by a semi-automated post-processing step, which includes removing duplicate images, as well as manual spot-checking for misclassified images.

Eventually, we identify 2,003 ad images and 7,432 non-ad images. To balance the positive and negative examples in our dataset so the classifier doesn't favor one class over another, we limited the number of non ad and ad images to 2,000.

4.4.2 Crawling with PERCIVAL

We found that traditional crawling was good enough to bootstrap the ad classification training process, but it has the fundamental disadvantage that for dynamically-updated elements, the meaningful content is often unavailable at the time of the screenshot, leading to screenshots filled with white-space.

More concretely, the *page load* event is not very reliable when it comes to loading iframes. Oftentimes when we take a screenshot of the webpage after the *page load* event, most of the iframes don't appear in the screenshots. Even if we wait a fixed amount of time before taking the screenshot, iframes constantly keep on refreshing, making it difficult to capture the rendered content within the iframe consistently.

To handle dynamically-updated data, we use PERCIVAL's browser architecture to read all image frames after the browser has decoded them, eliminating the race condition between the browser displaying the content and the screenshot we use to capture the image data. This way we are guaranteed to capture all the iframes that were rendered, independently of the time of rendering or refresh rate.

Instrumentation: Figure 5 shows how we use PERCIVAL's browser instrumentation to capture image data. Each encoded image invokes an instance of `DecodingImageGenerator` inside Blink, which in turn decodes the image using the relevant image decoder (PNG, GIFs, JPG, etc.). We use the buffer passed to the decoder to store pixels in a bitmap image file, which contains exactly what the render engine sees. Additionally, the browser passes this decoded image to PERCIVAL, which determines whether the image contains an ad. This way, every time the browser renders an image, we automatically store it and label it using our initially trained network, resulting in a much cleaner dataset.

Crawling: To crawl for ad and non-ad images, we run our PERCIVAL-based crawler with a browser automation tool called Puppeteer [18]. In each phase, the crawler visits the landing page of each Alexa top-1,000 websites, waits until `networkidle0` (when there are no more than 0 network connections for at least 500 ms) or 60 seconds. We do this to ensure that we give the ads enough time to load. Then our crawler finds all internal links embedded in the page. Afterwards, it visits 20 randomly selected links for each page, while waiting for `networkidle0` event or 60 seconds time out on each request.

In each phase, we crawl between 40,000 to 60,000 ad images. We then post process the images to remove duplicates, leaving around 15-20% of the collected results as useful. We crawl for a total of 8 phases, retraining PERCIVAL after each stage with the data obtained from the current and all the previous crawls. As before, we cap the number of non-ad images to the amount of ad image to ensure a balanced dataset.

This process was spread-out in time over 4 months, repeated every 15 days for a total of 8 phases, where each phase took 5 days. Our final dataset contains 63,000 unique images in total with a balanced split between positive and negative samples.

5 Evaluation

In this section, we evaluate the effectiveness of PERCIVAL in different settings. In Section 5.1, we validate PERCIVAL on an a random sample from the dataset created by Hussain et al. [38], which consists of 64,832 images annotated using Mechanical Turk workers. We train on our own dataset and test on the mentioned external dataset. In Section 5.2, we assess the accuracy of PERCIVAL by comparing against filter lists. In Section 5.3 we measure the accuracy of PERCIVAL on Facebook ads and sponsored content. In Section 5.4 we test PERCIVAL by fetching images from Google Image Search and report blocking results on various search queries. In Section 5.5, we analyze the performance of PERCIVAL on ads in languages other than English. In Section 5.6 we explore the features that our network learned by extracting the salience maps of the network. Finally, in Section 5.7 we present a run-time performance evaluation on both Chromium and Brave browsers equipped with PERCIVAL.

5.1 Accuracy Against an External Dataset

To validate our crawling procedure, training and network architecture, we tested PERCIVAL on the data set published

with Hussain et al. [38]. Our model was trained on the data set that we constructed by crawling top 500 Alexa sites. We gathered a total of 38,861 ad images, out of which we used 33,000 for training and 5,861 for validation. We then tested with a random sample of 5,024 ads from the dataset gathered by Hussain et al. [38]. Figure 8 shows the results of this experiment.

5.2 Accuracy Against EasyList

To evaluate whether PERCIVAL can be a viable shield against ads, we conduct a comparison against the most popular crowd-sourced ad blocking list, EasyList [7], currently being used by extensions such as Adblock Plus [1], uBlock Origin [23] and Ghostery [12].

Methodology: For our comparison we created two data sets, each one based on Alexa's top 500 news sites.

- We applied EasyList rules to select DOM elements that are potentially containers of ads (IFRAMEs, DIVs, etc.); we then captured screenshots of the contents of these elements, resulting in 6,930 images. We manually labelled them to identify the false positives.
- We used resource-blocking rules from EasyList to label all the images of each page according to their resource URL.

We refer to these two datasets as *screenshots* and *images*, respectively, and summarize dataset characteristics in Figure 6.

Performance: On our evaluation dataset, PERCIVAL is able to replicate the EasyList rules with accuracy 96.76%, precision 97.76% and recall 95.72% (Figure 7), illustrating a viable alternative to the manually-curated filter-lists.

5.3 Blocking Facebook Ads

Facebook's business model depends on serving first-party ads. Facebook obfuscates the "signatures" of ad elements (e.g. HTML classes and identifiers) used by filter lists to block ads. Over the years, the ad blocking community and Facebook have been playing a game of cat and mouse, where the ad blocking community identifies the ad signatures to block the ads and Facebook responds by changing the ad signatures or at times even the website [17]. Until recently, Adblock plus was able to block all ads and sponsored content for more than a year [2], however recently (late 2018) Facebook ads successfully managed to evade Adblock plus as the ad post code now looks identical to normal posts [10, 13] and rule-based filtering cannot detect these ads and sponsored content any more.

Sponsored posts and ads on Facebook generally look like ads and so far Facebook does not obfuscate the content of these posts, since Facebook is required to follow the rules on misleading advertising [10, 11]. Even though this requirement favors perceptual ad blockers over traditional ones, a lot of the content on Facebook is user-created which complicates the ability to model ad and non-ad content.

In this section, we assess the accuracy of PERCIVAL on blocking Facebook ads and sponsored content.

Methodology: To evaluate PERCIVAL's performance on Facebook, we browse Facebook with PERCIVAL for a period of 35 days using two non-burner accounts that have been in use for over 9 years. Every visit is a typical Facebook browsing session, where we browse through the feed, visit friends' profiles, and different pages of interest. For desktop computers two most popular places to serve ads is the right-side columns and within the feed (labelled sponsored) [9].

For our purposes, we consider content served in these elements as ad content and everything else as non-ad content. A true positive (TP) is defined as the number of ads correctly blocked, a true negative (TN) as the number of non-ads correctly rendered (or not blocked), a false positive (FP) as the number of non-ads incorrectly blocked and false negative (FN) is the number of ads PERCIVAL missed to block. For every session, we manually compute these numbers. Figure 10 shows the aggregate numbers from all the browsing sessions undertaken. Figure 12a shows PERCIVAL blocking right-side columns correctly and Figure 12b shows PERCIVAL blocking sponsored content within the feed correctly in addition to the right-side column ads.

Results: Our experiments show that PERCIVAL blocks ads on Facebook with a 92% accuracy and 0.784 and 0.7 as precision and recall, respectively. Figure 10 shows the complete results from this experiment. Even though we achieve the accuracy of 92%, there is a considerable number of false positives and false negatives, and as such, precision and recall are lower. The classifier always picks out the ads in the right-columns but struggles with the ads embedded in the feed. This is the source of majority of the false negatives. False positives come from high "ad intent" user-created content, as well as content created by brand or product pages on Facebook (Figure 11).

Dataset	Size	Matched rules
CSS rules	5,000	20.2%
Network	5,000	31.1%

Figure 6: Size of the data-set and percentage of ads identified by EasyList.

Images	Ads Identified	Accuracy	Precision	Recall
6,930	3466	96.76%	97.76%	95.72%

Figure 7: Summary of the results obtained by testing the dataset gathered using EasyList with PERCIVAL.

Size (images)	Acc.	Size	Avg. time	Precision	Recall	F1
5,024	0.877	1.9 MB	11 ms	0.815	0.976	0.888

Figure 8: PERCIVAL classification results on the dataset created by Hussain et al. [38]

Language	Images crawled	Ads Identified	Accuracy	Precision	Recall
Arabic	5008	2747	81.3%	0.833	0.825
Spanish	2539	309	95.1%	0.768	0.889
French	2414	366	93.9%	0.776	0.904
Korean	4296	506	76.9%	0.540	0.920
Chinese	2094	527	80.4%	0.742	0.715

Figure 9: Accuracy of PERCIVAL on ads in non-English languages. The second column represents the number of images we crawled, while the third column is the number of images that were identified as ads by a native speaker. The remaining columns indicate how well PERCIVAL is able to reproduce these labels.

Ads	Non-ads	Accuracy	FP	FN	TP	TN	Precision	Recall
354	1,830	92.0%	68	106	248	1,762	0.784	0.7

Figure 10: Online evaluation of Facebook ads and sponsored content.

(a) False Positive: This post was created by page owned by Dell Corp.

(b) False Negative: This post was part of the sponsored content in the news feed.

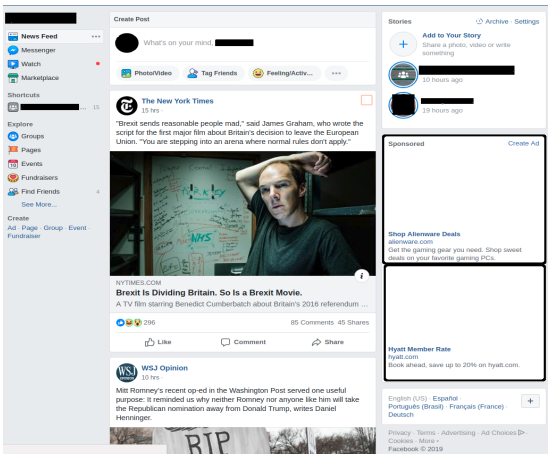
Figure 11: Examples of false positives and false negatives on Facebook.

5.4 Blocking Google Image Search Results

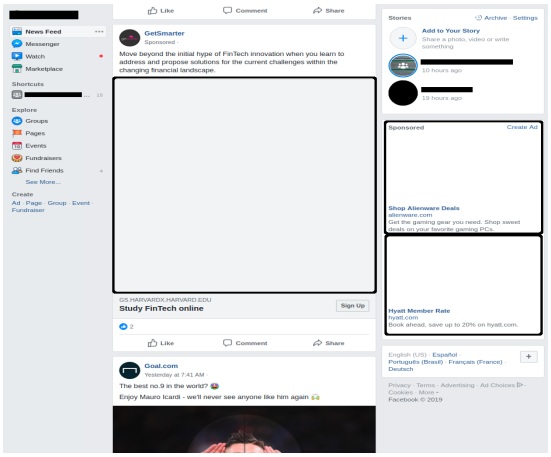
To improve our understanding of the misclassifications of PERCIVAL, we used Google Images as a way to fetch images from distributions that have high or low ad intent. For example, we fetched results with the query "Advertisement" and used PERCIVAL to classify and block images. As we can see in Figure 16b, out of the top 23 images, 20 of them were successfully blocked. Additionally, we tested with examples of low ad intent distribution we used the query "Obama"). As we can see in Figure 16a, out of 100 top images, only 12 images where blocked. We also searched for other keywords, such as "Pastry", "Shoes", "Coffee", etc. The detailed results are presented in Figure 13. As shown, PERCIVAL can identify a significant percentage of images on a highly ad-biased content.

5.5 Language-Agnostic Detection

We test PERCIVAL against images with language content different than the one we trained on. In particular, we source



(a) PERCIVAL blocking Facebook ads



(b) PERCIVAL blocking sponsored content embedded in the feed in addition to ads.

Figure 12: The screenshots show one of the author' Facebook home page accessed with PERCIVAL. The black rectangles are not part of the original screenshot.

Search query	Images blocked	Images rendered	FP	FN
Obama	12	88	12	0
Advertisement	96	4	0	4
Shoes	56	44	-	-
Pastry	14	86	-	-
Coffee	23	77	-	-
Detergent	85	15	10	6
iPhone	76	24	23	1

Figure 13: PERCIVAL blocking image search results. For each search we only consider the first 100 images returned ("-" represents cases where we were not able to determine whether the content served is ad or non-ad).

a data set of images in Arabic, Chinese, German, French, Korean and Spanish.

Crawling: To crawl for ad and non-ad images, we use ExpressVPN [8] to VPN into major world cities where the above mentioned languages are spoken. For instance, to crawl Korean ads, we VPN into two locations in Seoul.

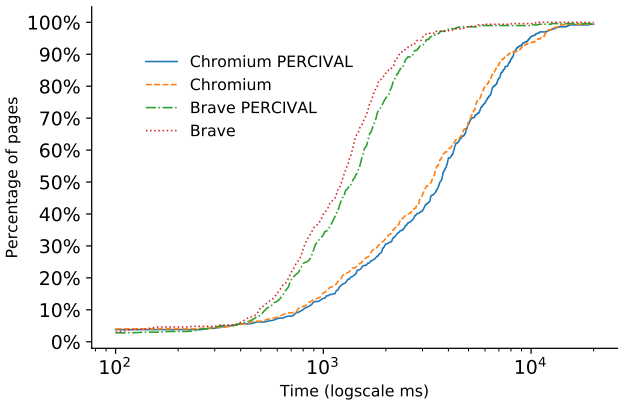


Figure 14: `Render` time evaluation in Chromium and Brave browser.

Baseline	Treatment	Overhead (%)	(ms)
Chromium	Chromium + PERCIVAL	4.55	178.23
Brave	Brave + PERCIVAL	19.07	281.85

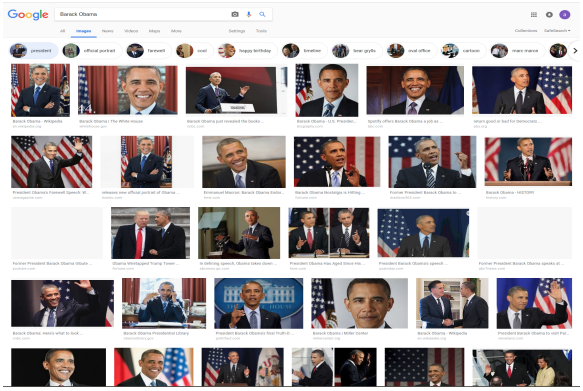
Figure 15: Performance evaluation of PERCIVAL on `Render` metric.

We then manually visit top 10 websites as mentioned in SimilarWeb [21] list. We engage with the ad-networks by clicking on ads, as well as closing the ads (icon at the top right corner of the ad) and then choosing random responses like content not relevant or ad seen multiple times. This is done to ensure we are served ads from the language of the region.

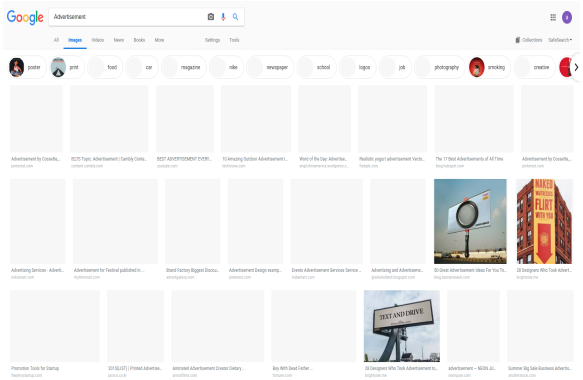
We then run PERCIVAL-based crawler with the browser automation tool Puppeteer [18]. Our crawler visits the landing page of each top 50 SimilarWeb websites for the given region, waits until `networkidle0` (when there are no more than 0 network connections for at least 500 ms) or 60 seconds. Then our crawler finds all internal links embedded in the page. Afterwards, it visits 10 randomly selected links for each page, while waiting for `networkidle0` event or 60 seconds time out on each request. As opposed to Section 4.4.2, we download every image frame to a single bucket.

Labeling: For each language, we crawl 2,000–6,000 images. We then hire a native speaker of the language under consideration and have them label the data crawled for that language. Afterwards, we test PERCIVAL with this labeled dataset to determine how accurately can PERCIVAL reproduce these human annotated labels. Figure 9 shows the detailed results from all languages we test on.

Results: Our experiments show that PERCIVAL can generalize to different languages with high accuracy (81.3% for Portuguese, 95.1% for Spanish, 93.9% for French) and moderately high precision and recall (0.833, 0.825 for Arabic, 0.768, 0.889 for Spanish, 0.776, 0.904 for French). This illustrates the out-of-the box benefit of using PERCIVAL for languages that have much lower coverage of EasyList rules, compared to the English ones. The model does not perform as well on Korean and Chinese datasets.



(a) Search results from searching for "Barack Obama" on Google images using PERCIVAL.

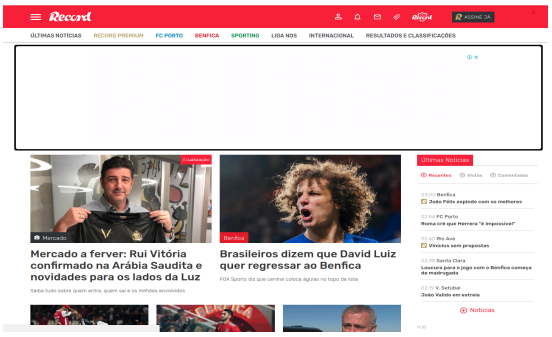


(b) Search results from searching for "Advertisement" on Google images, using PERCIVAL.

5.6 Salience Map of the CNN

While Convolutional Neural Networks [43] have superior performance compared to other computer vision techniques, they suffer from interpretability problems; it is difficult to understand why the network predicts a certain label. To visualize which segments of the image are influencing the classification decision, we used Grad-CAM [57] network salience mapping which allow us to highlight the important regions in the image that caused the prediction.

As we can see in Figure 4, our network is focusing on ad visual cues (*AdChoice* logo), when this is present (case (a)), also it follows the outlines of text (signifying existence of text between white space) or identifies features of the object of interest (wheels of a car). In case (d), it focuses on the clothing but also tracks some area from the sea. This is compatible with our intuition that ad images usually contain some text or visual cues but also common objects such as cars, clothes, etc.



(a) PERCIVAL results on `record.pt` (Portuguese language website).

(b) PERCIVAL results on `sohu.com` (Chinese language website).

Figure 17: PERCIVAL results on a few sites written in Non-English languages.

5.7 Runtime Performance Evaluation

We next evaluate the impact of PERCIVAL-based blocking on the performance of surrounding browser. This delay is a function to the number and complexity of the images on the page and the time the classifier takes to classify each of them. We measure the rendering time impact when we classify each image *synchronously*.

To evaluate the performance of our system, we used top 5,000 URLs from Alexa to test against Chromium compiled on Ubuntu Linux 16.04, with and without PERCIVAL activated. We also tested PERCIVAL in Brave, a privacy-oriented Chromium-based browser, which blocks ads using block lists by default. For each experiment we measured `render` time which is defined as the difference between `domComplete` and `domLoading` events timestamps. We conducted the evaluations sequentially on the same Amazon m5.large EC2 instance to avoid interference with other processes and make the comparison fair. Also, all the experiments where using `xvfb` for rendering, an in-memory display server which allowed us to run the tests without a display.

In our evaluation we show an increase of 178.23 ms of median render time when running PERCIVAL in the rendering critical path of Chromium and 281.85 ms when running inside Brave browser with ad-blocker and shields on. Figures 14 and 15 summarize the results.

6 Discussion

Deployment Concerns: In this paper, we primarily focus on deploying PERCIVAL as an in-browser blocker. There are significant advantages to implementing this in the code of the browser in C++ rather than as a browser extension.

However, we note that this is not the only way to use our approach. PERCIVAL can be used to build and enhance block lists for traditional ad blockers. For that we would need to set up a crawling infrastructure to find URLs and potentially DOM XPath expressions to block.

How to properly orchestrate crawling is not entirely clear: the simple approach of crawling a subset of the most popular sites such as those provided by Alexa will likely miss the long tail — unpopular sites that are not reached by such crawls but are reached by the long tail of users. However, we can still use such techniques to frequently update block lists automatically.

Yet a third approach is to collect URLs (and possibly XPath expressions) in the browser that are not already blocked by existing block lists, and then to crowd-source these from a variety of users.

However, all these techniques come with different user privacy trade-offs. Blocking too late in the pipeline (e.g. during rendering) provides context that opens the doors for machine learning based blocking techniques, sacrificing though the privacy of the user since the tracking mechanisms might have already run at this stage. On the other hand, blocking too early allows for higher privacy guarantees, but the blocking accuracy will depend on the effectiveness of the filter lists. We believe that a hybrid of rendering time and filter lists based blocking can help the creation of effective shields against ads that balance the trade-offs between accuracy and privacy.

Limitations: By testing PERCIVAL integrated into Chromium, we noticed the following limitations. Many ads consist of multiple elements, which contain images and text information layered together. PERCIVAL is positioned in the rendering engine, and therefore it has access to one image at a time. This leads to situations where we effectively block the image, but the text is left dangling. Unfortunately the nature of the in-rendering blocking does not allow post-rendering DOM tree manipulations, however, an effective solution to this would be to memorize the DOM element that contains the blocked image and filter it out on consecutive page visitations. Although this might provide an unsatisfying experience to the user, we argue that it is of the benefit of the user to eventually have a good ad blocking experience, even if this is happening on a second page visit.

Finally, as it has been demonstrated by Tramèr et al. [65], computer-vision based ad blockers suffer from adversarial attacks, where an attacker can perturb an image in order

Figure 18: Example of an ad. Ads usually contain (1) body text, (2) image text, (3) ad image.

to confuse the underlying classifier. Although this remains an open problem in the deep computer-vision community, there is a promising body of research [41, 42, 45, 46, 48] that is proposing solutions to mitigate the exploitability of such models. One way to partly address this in PERCIVAL is to incorporate back-propagation algorithm into the browser and retrain the model client side.

7 Related Work

Ad blocking: Online advertisement has contributed to the degradation of the user experience and has severely encroached on end-user privacy. The effectiveness of ad blocking has also been diminishing for a long time. To address these issues, browser extensions and ad-blocking browsers such as Brave and Opera [4, 15] have been developed that are blocking online ads and trackers. As of February 2017, 615 million devices had ad-blockers installed [16]. The significance and success of ad-blockers have turned the online advertising industry into anti-ad-blockers solutions and have created an arms-race between ad-blockers and advertisers [50]. As a response to the increasing popularity of ad-blockers, online advertisers have adopted anti-ad-blocking techniques as surveyed in [49].

Filter lists: Popular ad-blockers like, Adblock Plus [1], uBlock Origin [23], and Ghostery [12] are using a set of rules, called filter-list, to block resources that match a predefined crowd-sourced list of regular expressions (from lists like EasyList and EasyPrivacy). On top of that, CSS rules are applied, to prevent DOM elements that are potential containers of ads. These filter-lists are crowd-sourced and updated frequently to adjust on the non-stationary nature of the online ads [60]. For example, EasyList, the most popular filter-list, has a history of 9 years and contains more than 60.000 rules [67]. However, filter-list based solutions enable a continuous cat-and-mouse game: their maintenance cannot scale efficiently, as they depend on the human-annotator and they do not generalize to "unseen" examples.

Machine Learning Based Ad Blockers: Lately, techniques based on machine learning started to emerge. AdGraph [40] proposes a supervised machine-learning solution on the multi-layered graph representing the interaction of the HTML, HTTP and JavaScript of the page to identify ads and trackers. We see this work as complementary to ours since it exploits the latent interactions that happen inside the browser, to identify the resources of interest.

Perceptual Ad Blocking: Perceptual ad blocking is the idea of blocking ads based solely on their appearance; an example ad, highlighting some for the typical components, is shown in Figure 18. Storey et al. [60] uses the rendered image content to identify ads. More specifically, they use OCR and fuzzy image search techniques to identify *visual cues* such as ad disclosure markers or sponsored content links. Unlike PERCIVAL, this work assumes that the ad provider is complying with the legislation and is using visual cues like *AdChoices*. They also suggest using perceptual ad blocking for FB ads, although it appears that their technique relies chiefly on text matching and not training on a large number of images; further, they do not present an accuracy evaluation for their approach.

Sentinel system [58] proposes a solution based on convolutional neural networks (CNNs) to identify Facebook ads. This work is closer to our proposal; however, their model is not deployable in mobile devices or desktop computers because of its large size (based on YOLO [55] which is >200MB). We report on the performance of PERCIVAL on Facebook ads in Section 5.3.

Also, we would like to mention the work of [24, 38, 68], where they use deep neural networks to identify the represented signifiers in the Ad images. This is a promising direction in semantic and perceptual ad-blocking.

CNN image classification and model compression: Convolutional neural networks have been used for more than 20 years. Lecun et al. introduced CNN in [47]. Models like [44] have demonstrated that the power of Deep convolutional neural networks and since then they have become the state-of-the-art in the image classification tasks. More specifically, significant Deep Neural network architectures like [34, 59, 63] are being used currently in several commercial applications. However, these resource-hungry architectures cannot be deployed on devices with limited resources.

To address these issues, [39] proposed an architecture that can get AlexNet comparable accuracy with a model that is less than 0.5 MB. The authors introduced a novel network architecture combined with training tricks that helped them compress the same amount of knowledge on a significantly less size. Similarly, [36] has proposed models that are deployable on mobile devices making them appealing for commercial applications on devices with limited resources.

In [55], the authors demonstrated a real-time object detection system, with state-of-the-art accuracy. Although this model would be a great candidate to base PERCIVAL on, the model size (235 MB) is prohibitive for laptop or mobile browsers. Relevant to model compression is also the network distillation work from Hinton et al. [35].

However, these techniques do not come without drawbacks. In the seminal work Goodfellow et al. [30], demonstrated the

technique that came to be known as adversarial attacks of machine-learning models.

Adversarial attacks: One can generate adversarial examples that when "input to machine learning models causes the model to make a mistake" [30]. In computer-vision, researchers have demonstrated attacks that can cause prediction errors by near-imperceptible perturbations of the input image. This poses risks in a wide range of applications on which computer vision is a critical component (e.g. autonomous cars, surveillance systems) [51–53]. Similar attacks have been demonstrated in speech to text [27], malware detection [31] and reinforcement-learning [37].

In [65], Tramèr et al. analyzed the security risks of perceptual ad-blockers and showed that state-of-the-art perceptual ad-blockers are vulnerable to several attacks. However, most of the attacks presented work on element-based or frame-based perceptual ad-blockers. Element-based perceptual ad-blockers segment pages into HTML elements and assume a strict correspondence between elements in the DOM and how these are displayed in the webpage. PERCIVAL is not an element-based ad-blocker and operates on every image obtained during page execution regardless of which element displays it. Frame based perceptual ad-blockers map DOM to the rendered content. Attacks against these ad-blockers also involve DOM obfuscation and as such don't work with PERCIVAL. Attacks like Ad-Block Evasion for all website [65] where a publisher perturbs the full webpage using pixel level CSS techniques don't work with PERCIVAL either since PERCIVAL intercepts the image directly from the rendering pipeline. Page-based perceptual ad-blockers [20] work on images of the entire webpages and ignore DOM. Tramer et al. [65] demonstrated how these ad-blockers are suceptible to Cross-Boundary blocking, where a malicious user can upload adversarial content on part of the page which triggers the ad-blocker to block some other non-ad part. PERCIVAL makes each blocking decision independently so one blocking decision does not affect another.

Adversarial machine learning, on the other hand, is the most potent threat to perceptual ad-blocking. Advertisers can use the original neural network to create adversarial samples that fool the ad-blocker. To defend from adversarial attacks, a portfolio of techniques has been proposed [41, 42, 45, 46, 48], none of which seem to solve the adversarial-attacks problem.

8 Conclusions

The progress of computer-vision systems has allowed building systems that have a human-level performance for tasks that are solvable within a few seconds by humans experts, such as image classification. With these developments in machine learning, we have also observed a trend towards massive and resource-hungry models that require specialised hardware (like GPUs and TPUs) or privacy-invasive training procedures. Little effort has been invested in building models that are suitable for tasks on edge devices, like mobile phones and laptops. In our focus area, machine learning based ad blockers have made a recent appearance in the research community, and the heavily over-fitting filter list-based ad blockers are the dominant paradigm.

With PERCIVAL, we illustrate that it is possible to devise models that block ads, while rendering images inside the browser. Our implementation inside Chromium shows an rendering time overhead of 4.55%. This slowdown, although non-negligible, shows the feasibility of deploying traditionally heavyweight models (i.e. deep neural networks) inside the critical path of the rendering engine of a browser. We show that our perceptual ad-blocking model can replicate EasyList rules with an accuracy of 96.76%, making PERCIVAL into a viable and complementary ad blocking layer. Finally, we demonstrate off the shelf language-agnostic detection due to the fact that our models do not depend on textual information. Further, we also show that PERCIVAL is a compelling blocking mechanism for first-party Facebook sponsored content, for which traditional filter based solutions are less effective.

References

[1] Adblock Plus for Chrome support. https://adblockplus.org/.

[2] Advantage ABP successfully blocking ads on Facebook. https://adblockplus.org/blog/advantage-abp-successfully-blocking-ads-on-facebook.

[3] Annoying online ads do cost business. https://www.nngroup.com/articles/annoying-ads-cost-business/.

[4] Brave - Secure, Fast & Private Web Browser with Adblocker. https://brave.com/.

[5] Chromium Graphics. https://www.chromium.org/developers/design-documents/chromium-graphics.

[6] Coalition for better ads. https://www.betterads.org/research/.

[7] EasyList. https://easylist.to.

[8] ExpressVPN. https://www.expressvpn.com/.

[9] Facebook Ad placements. https://www.facebook.com/business/help/407108559393196.

[10] Facebook's Arms Race with Adblockers Continues to Escalate. https://motherboard.vice.com/en_us/article/7xydvx/facebooks-arms-race-with-adblockers-continues-to-escalate.

[11] False and deceptive display ads at yahoo's right media, 2009. http://www.benedelman.org/rightmedia-deception/#reg.

[12] Ghostery – Privacy Ad Blocker. https://www.ghostery.com/.

[13] I'm seeing "sponsored" posts on Facebook again. https://help.getadblock.com/support/solutions/articles/6000143593-i-m-seeing-sponsored-posts-on-facebook-again.

[14] Implement hide-if-contains-snippet. https://issues.adblockplus.org/ticket/7088/.

[15] Opera - Fast, secure, easy-to-use browser. https://www.opera.com/.

[16] Page Fair Ad Block Report. https://pagefair.com/blog/2017/adblockreport/.

[17] Ping pong with facebook. https://adblockplus.org/blog/ping-pong-with-facebook.

[18] Puppeteer: Headless Chrome Node API. https://github.com/GoogleChrome/puppeteer.

[19] Selenium: Web Browser Automation. https://www.seleniumhq.org.

[20] Sentinel: The artificial intelligenence ad detector.

[21] Similar Web. https://www.similarweb.com/.

[22] Skia Graphics Library. https://skia.org/.

[23] uBlock Origin. https://www.ublock.org/.

[24] Karuna Ahuja, Karan Sikka, Anirban Roy, and Ajay Divakaran. Understanding Visual Ads by Aligning Symbols and Objects using Co-Attention. 2018.

[25] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[26] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8), 2013.

[27] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *Proceedings - 2018 IEEE Symposium on Security and Privacy Workshops, SPW 2018*, 2018.

[28] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, 2016. ACM.

[29] G. Storey, D. Reisman, J. Mayer and A. Narayanan. Perceptual Ad Highlighter. https://chrome.google.com/webstore/detail/perceptual-ad-highlighter/mahgiflleahghaapkboihnbhdplhnchp, 2017.

[30] Ian Goodfellow, Nicolas Papernot, Sandy Huang, Yan Duan, Pieter Abbeel, and Jack Clark. Attacking Machine Learning with Adversarial Examples. *OpenAI. https://blog. openai. com/adversarial-example-research*, 2017.

[31] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*. Springer, 2017.

[32] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. An automated approach for complementing ad blockers' blacklists. *Proceedings on Privacy Enhancing Technologies*, 2015(2), 2015.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[35] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. 2015.

[36] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017.

[37] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[38] Zaeem Hussain, Christopher Thomas, Mingda Zhang, Zuha Agha, Xiaozhong Zhang, Nathan Ong, Keren Ye, and Adriana Kovashka. Automatic understanding of image and video advertisements. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January(14):1100–1110, 2017.

[39] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. 2016.

[40] Umar Iqbal, Zubair Shafiq, Peter Snyder, Shitong Zhu, Zhiyun Qian, and Benjamin Livshits. Adgraph: A machine learning approach to automatic and effective adblocking. *CoRR*, abs/1805.09155, 2018.

[41] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial Logit Pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[42] J Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 1(2), 2017.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, USA, 2012. Curran Associates Inc.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

[45] Alex Kurakin, Dan Boneh, Florian Tramèr, Ian Goodfellow, Nicolas Papernot, and Patrick McDaniel. Ensemble Adversarial Training: Attacks and Defenses. 2018.

[46] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[47] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

[48] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[49] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E Powles, E D Cristofaro, Hamed Haddadi, and Steven J Murdoch. Adblocking and counter blocking: A slice of the arms race. In *CoRR*, volume 16. USENIX Foci, 2016.

[50] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E Powles, Emiliano De Cristofaro, Hamed Haddadi, and Steven J Murdoch. Ad-Blocking and Counter Blocking: A Slice of the Arms Race. *Usenix Foci*, 2016.

[51] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. 2016.

[52] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. 2016.

[53] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, 2016.

[54] Enric Pujol, Tu Berlin, Oliver Hohlfeld, Anja Feldmann, and Tu Berlin. Annoyed users: Ads and ad-block usage in the wild.

[55] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[56] Sam Tolomei. Shrinking APKs, growing installs. https://medium.com/googleplaydev/shrinking-apks-growing-installs-5d3fcba23ce2.

[57] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[58] Adblock Sentinel. Adblock Plus, Sentinel, 2018.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[60] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. The Future of Ad Blocking: An Analytical Framework and New Techniques. 2017.

[61] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. The Future of Ad Blocking: An Analytical Framework and New Techniques. 2017.

[62] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[64] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *NIPS*, 2013.

[65] Florian Tramèr, Pascal Dupré, and Giancarlo Pellegrino. Ad-versarial : Defeating Perceptual Ad-Blocking.

[66] Antoine Vastel, Peter Snyder, and Benjamin Livshits. Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking. *CoRR*, abs/1810.09160, 2018.

[67] Antoine Vastel, Peter Snyder, and Benjamin Livshits. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. *arXiv preprint arXiv:1810.09160*, 2018.

[68] Keren Ye and Adriana Kovashka. ADVISE: symbolism and external knowledge for decoding advertisements. *CoRR*, abs/1711.06666, 2017.