University of Stuttgart

Institute for Parallel and Distributed Systems - AC

Universitätsstraße 32
70569 Stuttgart

**Bachelor Thesis**

# Icon Semantic Classification

Jan Knecht

| | |
|---|---|
| **Study program:** | B.Sc. Informatik |
| **1. Examiner:** | Prof. Dr. Steffen Staab |
| **2. Examiner:** | |
| **Advisors:** | Ramin Hedeshy, M.Sc., Dr. Raphael Menges |
| **start date:** | 01.02.2022 |
| **end date:** | 01.08.2022 |

## Abstract

Icons are a key building block of many modern day applications and websites. They make user interfaces more engaging and aesthetically pleasing while delivering meaning to the user in a compact and glanceable way. However, icons require special attention from designers and developers to make them accessible to people who rely on screen readers. In order for screen readers to recognize and read out icons to the user, they need special accessibility annotations that provide a textual description of an icon's purpose and function. As various studies show, such annotations are missing in many cases. To solve this problem, we propose a semantic icon classifier that can be used to predict the semantic class of an icon based on its pixel values. The predicted label can then be further used to improve accessibility of icons. While there has been various work on classifying icons by their semantics on mobile platforms, less attention has been paid to the web, despite possible differences in the appearance and variety of icons. By extending a dataset of icons found in mobile applications with web icons of open-source icon sets, icon databases and websites, we improve the accuracy of a semantic icon classifier on a real-world task. Furthermore, we propose a, to the best of our knowledge, novel classification approach to automatically generate semantic labels for compound icons. This allows us to predict more detailed classes for each icon. We demonstrate the real-world practicality of the system by implementing a browser extension that automatically attaches missing accessibility annotations to icons.

## Kurzfassung

Icons sind ein wichtiger Baustein vieler moderner Anwendungen und Websites. Sie machen Benutzeroberflächen einladender und ästhetisch ansprechender und vermitteln dem Benutzer auf kompakte und überschaubare Weise Informationen. Icons erfordern jedoch besondere Aufmerksamkeit von Designern und Entwicklern, um sie für Menschen zugänglich zu machen, die auf Screenreader angewiesen sind. Damit Screenreader Icons erkennen und dem Benutzer vorlesen können, benötigen Icons spezielle Annotationen, die eine textuelle Beschreibung des Zwecks und der Funktion des Icons enthalten. Wie verschiedene Studien zeigen, fehlen solche Anmerkungen jedoch in vielen Fällen. Um dieses Problem zu lösen, schlagen wir einen semantischen Icon Classifier vor, mit dem die semantische Klasse eines Icons auf der Grundlage seiner Pixelwerte vorhergesagt werden kann. Das vorhergesagte Label kann dann weiter verwendet werden, um die Barrierefreiheit von Icons zu verbessern. Während es verschiedene Forschungsarbeiten zur Klassifizierung von Icons anhand ihrer Semantik auf mobilen Plattformen gibt, wurde dem Web weniger Aufmerksamkeit geschenkt, trotz möglicher Unterschiede im Aussehen und der Vielfalt von Icons. Durch die Erweiterung eines Datensatzes von Icons aus mobilen Anwendungen mit Web-Icons aus Open Source Iconsammlungen, Icon-Datenbanken und Websites verbessern wir die Genauigkeit eines semantischen Icon Classifiers. Darüber hinaus schlagen wir einen, nach unserem Wissensstand, neuartigen Klassifikationsansatz vor, um automatisch semantische Bezeichnungen für zusammengesetzte Icons zu generieren. Dies ermöglicht uns, detailliertere Klassen für jedes Icon vorherzusagen. Wir demonstrieren die Praxistauglichkeit des Systems, indem wir eine Browsererweiterung implementieren, die fehlende Annotationen zur Barrierefreiheit automatisch an Icons anfügt.

# Contents

# 1 Introduction

Icons play a vital role in modern graphical user interfaces. For most applications and websites, icon-based buttons and links are especially important. They can be used to expose features to the user in a compact, memorable, and visually appealing manner [BSG89; Git86; MCD99]. In contrast to buttons that rely on text to carry meaning to the user, the semantics of an icon button are solely perceived by the shape and color of the icon. This naturally poses a challenge to users that have problems with their eyesight and puts them at a major disadvantage at everyday tasks. According to the World Health Organization[1] over 2 billion people worldwide have some sort of vision impairment. In severe cases, such as blindness, users can use screen readers to interact with the digital world. Screen Readers are computer programs that read the screen's content to the user and therefore enable them to interact with a computer in a non-visual way [2]. Unfortunately, icons and images pose a substantial problem for screen readers and other similar assistive technology because there is no textual description that can be read to the user.

To make icons accessible for users with vision impairments, they can be annotated by developers with additional metadata that describes their appearance and semantics. Screen readers can then detect this metadata and read aloud the provided metadata description of the icon, as if it was text. Unfortunately, many websites and applications are missing such annotations. In a large-scale analysis, Ross et al. [RZFW18] show that as of 2018 only 25.9% of 5,753 examined Android applications had 90% or more of their image-based buttons labeled. Conversely, large parts of most apps remain inaccessible to users that rely on screen readers or other forms of assistive tools that require additional information via metadata. This also applies to the web, especially as today, lines between native and web applications are blurring.

To solve this problem, the goal of this thesis is to develop an algorithm that can assign semantic classes to common types of icons found on the web. These classes can then be used to improve the accessibility of non- or insufficiently annotated icons.

While classifying icons using machine learning is not a novel idea, existing work focuses on mobile platforms[CCX+20; LCS+18; ZXC21] and leaves out possible differences between icon appearance on the web and mobile applications. At first glance, icon classification for mobile applications appears to be similar to icon classification on the web, but we hypothesize that web icons are more diverse than their mobile counterparts. Smartphone operating system vendors like Google and Apple strongly encourage the usage of a certain icon style in their design guidelines, for example Google Material Design Icons for Android[3] or SF Symbols for iOS[4]. These guidelines exist to offer the user a consistent user experience across apps and coherent visuals between the UI of the

[1] https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment

[2] https://developer.mozilla.org/en-US/docs/Glossary/Screen_reader

[3] https://material.io/

[4] https://developer.apple.com/design/human-interface-guidelines/

operating system and that of applications. On the web, there is no such central entity regulating or recommending a certain style. Most sites vary drastically in design, and so do their icons. Figure 1.1 shows icons of the semantic class “share” from various open-source web icon libraries. The visual appearance of the icons varies greatly, but they all share a common meaning.

**Figure 1.1:** Icons of the semantic class “share” from various open source libraries.

To achieve our goals, we base our work on the findings of Liu et al.[LCS+18], who introduce a successful approach to classifying icons on mobile platforms. We lay our focus on the web platform and aim to improve the overall quantity and accuracy of predicted icon semantic classes.

We structure our methodology into four distinct steps:

1. Create a dataset of icons with semantic annotations from various open-source icon libraries, databases and websites (Chapter 4).
2. Analyze the appearance and visual identity of icons in the dataset using a deep learning approach proposed by Lagunas et al. [LGG19]. We aim to find possible discrepancies in style similarity between icons found in mobile and web applications (Section 4.4) indicating that a special treatment of icons on the web is required.
3. Train and evaluate a classifier in form of a convolutional neural network on the collected dataset of icons (Chapter 5).
4. Automate icon labeling on the web by developing a browser extension that uses the icon classifier to automatically attach missing accessibility metadata to inaccessible icon elements. Thus, making previously inaccessible parts of the web accessible to users who rely on screen readers (Chapter 6).

Whereby, the main contribution of this thesis is threefold:

1. We improve the classifier accuracy on icons on the web by training it on an extended dataset compared to the dataset containing mobile icons by Liu et al.
2. To the best of our knowledge, this thesis is the first study proposing a novel method for classifying the semantic class of compound icons. This allows us to assign more detailed semantic classes to icons compared to the approach by Liu et al.

3. We demonstrate the real-world practicality of the classifier as a supportive tool for screen readers on the web via a browser extension.

# 2 Related Work

There has been various work on using machine learning for user interface semantic understanding, including classification and object detection of icons. However, most research is limited to mobile applications.

Liu et al. [LCS+18] (2018) propose a method for generating semantic annotations for mobile Android applications. By using the view hierarchy of an application and an accompanying screenshot, they can detect 25 distinct user interface components types, including icons, buttons, maps and images. For the icon component type, they develop a classifier based on a convolutional neural network that can detect an icon's semantic class out of 99 in total. The authors build a dataset of icons by annotating the Rico[DHF+17] dataset that contains design resources from over 9700 Android applications. The trained model achieves a validation accuracy of over 94%. Additionally, Liu et al. carry out an anomaly detection to detect images that are not icons with an accuracy of 90%.

Zang et al.[ZXC21] (2021) examine the benefits of using a multimodal classification and object detection approach for icons in Android applications over a vision only based approach as proposed, e. g., by Liu et al. Zang et al. implement and evaluate multiple classification and object detection models and find that the multimodal approach including the view hierarchy outperforms a pixel-only approach.

A different system to generating icon labels has been proposed by Chen et al.[CCX+20] (2020). Instead of relying on classification to determine the class of an icon, they generate a textual label using an encoder-decoder neural network. In a study of 10,408 applications in the Google Play Store, they show that 77% of applications have at least some missing accessibility annotations. Chen et al. find that the automatically generated textual labels are superior to those of Android application developers.

We identify related work in the areas of voice control and application security as use cases for semantic icon classification that go beyond accessibility.

IconNet [BS] (2021) by Google Research is a vision-based object detection model for detecting and labeling icons in mobile Android applications. It is part of Google's "Voice Access" accessibility service that allows users to control their smartphone by voice commands only. It can locate, detect and classify up to 31 different types of icons in a screenshot and is optimized to run on mobile devices. IconNet operates solely on the pixel values of a screenshot, i. e., does not need a view hierarchy. The system achieves a mean average precision of 94.2% and can run at a frame rate of 9Hz on a mid-range smartphone.

IconIntent [XWC+19] (2019) is an app analysis framework proposed by Xiao et al. IconIntent aims to determine the intent of UI widgets in mobile Android applications in a privacy context. A widget can be text (text-icon) or image-based (object-icon). The widget's intention is defined as one of eight labels mapped to sensitive data usage like camera or microphone access. If there is a discrepancy between the detected intent of the widget and the app's behavior, the app can

be flagged for further inspection. IconIntent uses the app's view hierarchy to detect icons with program static analysis. It can classify icons into 8 sensitive data categories with a precision of 88.4%. IconIntent uses the FAST algorithm for classification and does not rely on a neural network. DeepIntent[XYX+19] (2019) by Xi et al. is an improvement over the IconIntent Framework that uses neural networks for detecting the icon's intent. DeepIntent achieves superior results compared to IconIntent with a 19.3% relative improvement in permission detection.

While all of the above work focuses on mobile platforms, icon classification with focus on the web has been limited. Auto-Icon [FMY+21] (2021) by Feng et al. is a system aimed to ease website development workflow by automatically generating icon fonts from images. The authors build their dataset by obtaining icons from icon sharing websites. As part of a greater system, including converting the images to vectors and detecting their color, icon images are classified into one of 100 classes by a convolutional neural network. The resulting class is then used as a label for the icon in the font. The convolutional neural network achieves an accuracy of 88%. The approach of Feng et al. is similar to the methodology proposed in this paper. However, there are significant differences. While Feng et al. aim to improve accessibility during the development phase of an application and have the primary goal of speeding up the development process, this work focuses on improving accessibility of icons in existing applications that are not or only partially annotated with accessibility labels. We put this difference into practice as described in Chapter 6.

# 3 Background

## 3.1 Neural Networks

Neural networks are a powerful and widely used machine learning tool that have found large success in many fields of computer science including classification and object detection of icons [CCX+20; FMY+21; LCS+18; XYX+19].

The following sections about artificial neurons and neural networks are primarily based on the University of Stuttgart "Grundlagen der künstlichen Intelligenz WS19/20" course script by Marc Toussaint which cites "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig[RN09] as its main literature source.

### 3.1.1 Artificial Neurons

Artificial neurons are inspired by biological neurons, found for example in the human brain or nervous system [ON15; RN09]. A single artificial neuron is the most basic form of an artificial neural network. It takes one or more input signals from other neurons and produces an output signal. The strength of the output signal is based on a weighted sum of all its inputs and a constant bias. Artificial neurons therefore compute a linear operation on their input signals. To allow neurons to compute non-linear functions, a non-linear activation function $\sigma$ is applied to the output signal [RN09]. The output signal $y$ of an artificial neuron can be written as:

$$y = \sigma(\sum_{i=0}^{n} w_i x_i + b) \tag{3.1}$$

where $x$ is a vector of input signals, $w$ a vector of weights for the input signal and $b$ a constant bias.

Today, a common activation function is the rectified linear unit (ReLU)[GWK+18]:

$$\sigma(z) = \max(0, z) \tag{3.2}$$

Other activation functions include the leaky ReLU $\sigma(z) = \max(0.01z, z)$, sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$ or tanh.

### 3.1.2 Feedforward Neural Networks

A feedforward neural network is a sequence of layers of artificial neurons. In this sequence, every neuron from one layer is connected to every neuron of the next layer, i. e., one layer is "fed" into the next. These layers are also referred to as fully connected layers (FC layers). A feedforward neural network receives an input, transforms it through the series of layers, and produces an output.

Mathematically, a neural network with $L$ layers that each contain $h_l$ neurons, is a function $\mathbb{R}^{h_0} \mapsto \mathbb{R}^{h_L}$ with the parameter $\beta = W_{1:L}, b_{1:L}$ where $\beta$ is the collection of all weights $W_l \in \mathbb{R}^{h_l \times h_{l-1}}$ and biases $b_l \in \mathbb{R}^{h_l}$ of the neurons in the network. That is, $W_l$ is the weight matrix for the $l$-th layer and $b_l$ is the bias vector for the $l$-th layer.

Instead of describing the inputs and outputs of a single layer in terms of individual neurons, we can combine these calculations into a matrix multiplication. We can describe the input of a layer of artificial neurons as a vector $x$. It contains either the network's input or the activations of the previous layer. The output of the current layer can then be computed by multiplying the input vector $x$ with the weight matrix $W$ that contains the pairwise connection weights between the neurons in the previous and current layer. Subsequently, the bias vector $b$ containing all biases is added to the result and an activation function $\sigma$ is applied element-wise. The final output vector can then be passed as input to the next layer.

The first layer of the network is the input layer. Instead of being connected to other neurons in a previous layer, it receives the input of the network. The next layer then takes these inputs and transform them using its weights, biases and the activation function. The output of the network at this stage can be written as:

$$f_\beta(x) = \sigma(W_1 x + b_1) \tag{3.3}$$

where $x$ is the input vector to the neural network. The next layer of the network then repeats this procedure. It transforms the output of the previous layer using its weights, biases and activation function. The output now is:

$$f_\beta(x) = \sigma(W_2 \sigma(W_1 x + b_1) + b_1) \tag{3.4}$$

This scheme continues until the last layer $L$ is reached. This layer is also referred to as the output layer, as its output is simultaneously the output of the whole network. In total, the network can be described as the deeply nested function:

$$f_\beta(x) = \sigma(W_L \sigma(W_{L-1}(\dots \sigma(W_1 x + b_1) \dots) + b_{L-1}) + b_L) \tag{3.5}$$

The overall process of data getting transformed by flowing through the different layers of the network is referred to as forward propagation.

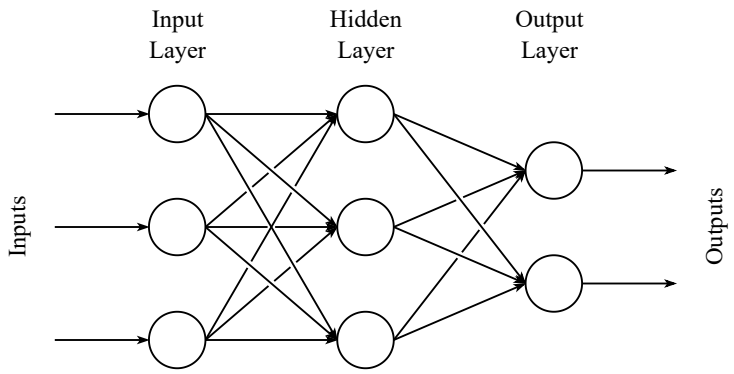


**Figure 3.1:** Illustration of a feed-forward neural network with three layers.

Layers between the input and output layer are referred to as hidden layers. Neural networks with many hidden layers are called deep neural networks.

Figure 3.1 shows a feed-forward neural network containing three FC layers. It takes a three-dimensional vector as input and outputs a two-dimensional vector, i. e., it computes a function $\mathbb{R}^3 \mapsto \mathbb{R}^2$. Each neuron of the input layer is connected to every neuron in the second layer. This process is repeated for the output layer.

### 3.1.3 Training

For the neural network to output meaningful results, its weights and biases have to be adjusted so its output matches a desired output. This process is called training.

The collection of weights and biases $\beta$ is obtained by minimizing a loss function $\ell$ on a Dataset $D = \{(x_i, y_i)\}_{i=1}^n$.

$$\beta^* = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(f_\beta(x_i), y_i) \tag{3.6}$$

The loss function $\ell$ is a measure of the difference between the output of the network $f_\beta(x_i)$ and the desired output $y_i$ for a data point $x_i$ of the dataset. The closer the network output is to the desired output, the smaller the loss. A loss of 0 for a specific data point means that the network outputs exactly the desired output $y_i$. Common loss functions are, e. g., mean squared error (MSE) for regression or categorical cross entropy for classification tasks.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(f_\beta(x_i) - y_i\right)^2 \tag{3.7}$$

This minimization of the loss function, i. e., the optimization of weights and biases, can be achieved using the gradient descent algorithm. Gradient descent is an iterative method. During gradient descent, the loss and its gradient are computed for the whole dataset. Then the weights and biases

are updated along the negative direction of the gradient. This causes the loss to gradually improve, i. e., reduce. This process is repeated until no further improvements can be made. The network has then converged to a local minimum of the loss function [Rud16].

To efficiently obtain the gradient of the loss, the backpropagation algorithm is used. It works in two steps: First, a data point is passed as input to the neural network and forward propagation is performed. Based on the resulting network output, the loss is calculated. Now, the partial derivative of the loss with respects to all weights and biases of the network is computed iteratively and backwards throughout the network. Because the neural network is a deeply nested function, the chain rule can be recursively applied to calculate each partial derivative for every weight and bias. This is repeated for every data point in the dataset. The final gradient is then obtained by summing the gradients of the loss function for each data point.

For large datasets, computing the gradient for all data points is computationally very expensive. Therefore, the dataset is split into smaller mini batches. After each mini batch, the gradient is computed using backpropagation and the weights and biases are adjusted accordingly. This procedure is also referred to as stochastic gradient descent[Rud16].

### 3.1.4 Convolutional Neural Networks

Convolutional neural networks (CNN) are special types of neural networks that focus on image processing. CNNs deal with the fact that regular neural networks featuring only FC layers are not feasible when using large images as input [ON15].

Images as inputs for neural networks are tensors with the three dimensions. Two spatial dimensions, the width and height of the image, and a third depth dimension that represents the number of channels in the image. For example, one channel for grayscale images, three for RGB images and four for RGBA images. With traditional FC layers, the number of weights and biases required to process an image is large [AMA17; ON15]. Every neuron in a layer is connected to every neuron in the next layer. For RGB images with a pixel size of $256 \times 256$ this already leads to every neuron in the first layer having over $256 \cdot 256 \cdot 3 = 196608$ connections. The result is a network with an overall high parameter count which requires a large amount of computing resource. Furthermore, the network is prone to overfitting[ON15]. CNNs reduce the parameter count greatly by embracing the properties of images as input data. This is achieved by introducing two additional layer types: convolutional and pooling layers.

Convolutional layers are layers that perform convolution on their input. In computer vision, convolutions are used to apply filters like sharpening or edge detection to images. This allows to extract abstract features from images that are not directly visible in the original but are useful for further processing in deeper layers of the network [AMA17]. Convolutional layers learn such filters during the training process to extract meaningful features from the input image. The learned filters that are convolved with the input image are typically small in the spatial dimensions, but extend throughout the whole image depth. For example, the MobileNetV2[SHZ+18] network used in this thesis uses filters with the spatial dimensions $3 \times 3$. To convolve these small filters with the large input image, they are moved across the image input step by step. The filter gets aligned with every pixel in the input image and convolution is performed. Each convolution of the filter with the input image results in an activation value. All the activation values combined result in a two-dimensional activation map that represents convolution across the input image as a whole [CCX+20; ON15].

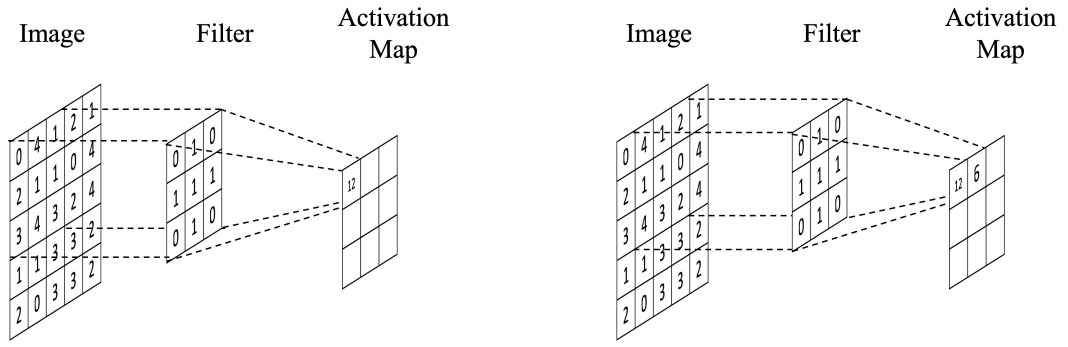


**Figure 3.2:** Illustration of the convolution operations within a convolutional layer.

Figure 3.2 shows an illustration of the convolution operations within a convolutional layer. The input image with shape $5 \times 5$ is shown on the left, overlaid by a convolution filter of size $3 \times 3$. First, the filter is aligned with the top left of the input image. The filter values are multiplied with the input image, and the results are summed up. The sum is written to the corresponding location in the activation map visible on the right. The next entry in the activation map is then computed by moving the filter to the right and repeating the same calculation. This process is repeated until every pixel is in the input image is convolved with the filter. The output of the convolution layer, the activation map, is a matrix with the shape $3 \times 3$.

A single convolutional layer can have many filters. If this is the case, each of the filters is convolved with the input image separately and the resulting activation maps are stacked to create a three-dimensional tensor. This tensor can then be passed to the next convolutional or pooling layer.

Convolutional layers have two additional hyperparameters, i. e., parameters, that are not learned by the network: stride and padding. Stride indicates the number of pixels to move the filter across the input image after every convolution. Padding is a technique that adds, e. g., zeros, to the outside of the input image to prevent the filters from going outside the image. Stride and padding both affect the layer's output size [ON15].

Just like FC layers, a convolutional layer can also be seen as an arrangement of neurons [ON15]. Every neuron corresponds to a value in the activation map. However, unlike FC layers, every neuron is not connected to all neurons in the previous layer but only a small subset. This subset is referred to as the neuron's receptive field. The receptive field is the area of the input image that is convolved with the filter to result in the activation value represented by that neuron. In addition to not being connected to all previous layer neurons, neurons in a convolutional layer share weights with their neighbors (parameter sharing)[ON15]. This is because the weights of the connection of the neurons represent the filter that is used to convolve the input image. Filters are the same regardless of pixel location, i. e., they are spatially invariant. This save the network from having to learn numerous weights for each neuron.

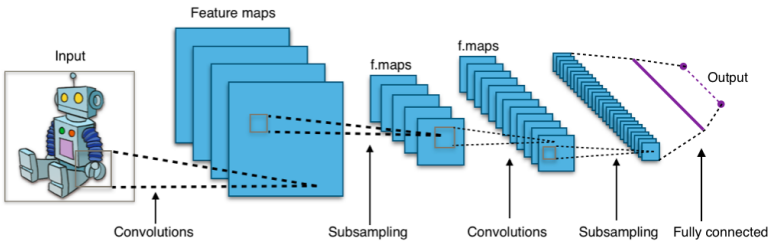


**Figure 3.3:** A typical convolutional Neural Network. Illustration by "Aphex34", distributed under a CC BY-SA 4.0 License [Aph15].

The second newly introduced layer type are pooling layers. Pooling layers are layers that downsample across the spatial dimension (width, height) of their input to reduce the computational complexity in following layer [CCX+20; ON15]. The pooling layer looks at a section of its input tensor, e. g., $2 \times 2$ square, and extracts dominant features by taking the maximum of the values. Just like convolutional layers the sliding window is then moved by the stride size and the process is repeated. Other variants like average instead of max pooling are possible. Pooling layers have no learnable parameters.

A typical CNN architecture for classification tasks consists of multiple stacked convolutional and pooling layers on top of one or more FC layers [ON15]. The convolution and pooling layers cause the network to learn to extract meaningful features in the input image while gradually reducing its size. Figure Figure 3.3 shows a CNN with two convolutional and pooling layers and a final fully connected layer. After the input layer takes the pixel values of the image as input, the network applies convolution to the image. The output size of the first convolutional layer is then downsampled using a pooling layer. Another convolution and downsampling layer follow, further reducing the size of the image. Finally, a flatten operation is applied, which transforms the multidimensional output of the last pooling layer into a one-dimensional vector that is passed to the final FC layers.

Today, CNNs are used in a variety of applications and use-cases and are one of the most popular machine learning approaches to image classification[GWK+18; ON15]. CNNs have been used successfully for image classification and object detection in fields like user interface semantic understanding[CCX+20; LCS+18], medicine[YNDT18], and audio and speech recognition[AMJ+14].

## 3.2 Web Accessibility

The Cambridge Dictionary defines accessibility as "the quality or characteristic of something that makes it possible to approach, enter, or use it"[Cam22]. According to the World Wide Web Consortium[1] (W3C) in the context of user interfaces and the World Wide Web, the focus of accessibility lies on making a product perceivable, understandable, navigatable and interactable for people with disabilities[2]. There is a wide variety of disabilities that have to be taken to account for, reaching from auditory and visual to cognitive and motor impairments.

[1] https://www.w3.org/

[2] https://www.w3.org/WAI/fundamentals/accessibility-intro/

The W3C Web Accessibility Initiative (WAI) publishes the Web Content Accessibility Guidelines (WCAG)[3]. The WCAG are a technical standard that aim to help developers make web content more accessible to users with disabilities. The guidelines are divided into the four main principles "perceivable", "operable", "understandable", and "robust". Regarding icons and images, the WAI states the general rules "provide text alternatives for non-text content" and "provide captions and other alternatives for multimedia". For icons, i. e., "Functional Images"), this means that they should either have an accompanying text that describes the icon's function and meaning or have an alternative text as an accessibility metadata attribute (Guideline 1.1.1).

In addition to WCAG the WAI also publishes the Accessible Rich Internet Applications (ARIA) standard[4]. WCAI-ARIA is a framework that help developers to meet WCAG in modern and highly interactive web applications. WCAI-ARIA brings a set of roles and attributes that can be attached to elements in the HTML documents. Browsers can then pick up these roles and attributes to compute an accessibility name for an element. These WCAI-ARIA attributes have the highest level of precedence when calculating accessibility names for elements.

For this thesis, the attribute aria-label is particularly important. The aria-label attribute is used to provide a label for an element without having to make the label visible to the user. The label should accurately describe the purpose of the element. An example use-case of aria-label is a button containing only an icon but no text. The aria-label attribute can be used to provide a label for the button.

[3] https://www.w3.org/WAI/standards-guidelines/wcag/
[4] https://www.w3.org/WAI/standards-guidelines/aria/

# 4 Dataset

We aim to create a Dataset $D = \{(x_i, y_{i,1}, y_{i,2})\}_{i=1}^{n}$ of icons where $x_i$ is an image of an icon, $y_{i,1}$ is the icons primary label and $y_{i,2}$ the icons secondary label as described in Section 4.1. The dataset should be representative of icons that are used on real-world sites on the web today.

Each icon $x_i$ is in the resolution $96 \times 96$ pixels and in the grayscale color space. It may have any background or foreground color with sufficient contrast, and may be cropped from a larger screenshots. Each icon is centered in the image and do not contain excessive padding or whitespace.

## 4.1 Semantic Classes

In order to create a dataset of icons and their respective semantic classes, we first have to understand how the semantics of icons can be defined, modeled and obtained.

Every icon belongs to a visual and a semantic class. While the visual class describes the icon's appearance, shape and symbols, the semantic class refers to the icon's meaning or function, i. e., what it is intended to represent [MCD99]. As Gittins states, the semantic classes of icons in graphical user interfaces are objects within a computer system, e. g., data or processes [Git86; GSF01]. More specifically, we refer to them as either an object (data), for example "email", or an action (process), for example "add".

The semantic class of an icon can be different from its visual class. For example, an icon depicting a paperclip belongs to a visual class of "paperclip" but to a semantic class of "attachment". This concept is also referred to as semantic distance [MCD99]. Semantic distance is a measure of the closeness of the relationship between an icon and its meaning [MCD99]. Mcdougall et al.[MCD99] note that semantic distance can be best understood as a continuum.

Related to the concept of semantic distance, multiple taxonomies have been proposed to group icons into categories. Generally, icons can be either representational icons, abstract icons or arbitrary icons [BSG89; GSF01]. Figure 4.1 shows examples of each of the three icon types. The left-most icon in Figure 4.1 is a representational icon. Representational icons, like images, visually map closely to the object they represent. The icons are a direct representation of their semantics. The semantic distance is small. Arbitrary icons on the other hand lack a clear representational reference to their meaning. An extreme example of an abstract icon is the biohazard warning sign [MC04] shown on the right of Figure 4.1. It neither shares a representation nor related concept with its meaning, but features a visually completely arbitrary appearance. The semantic distance is high. Abstract or implied icons exhibit characteristics from representational and arbitrary icons, and may be constructed out of either. They visualize a close concept to their meaning, but are not representational. Figure 4.1 shows an example of semi-abstract icons in the center. The icon shows an arrow pointing towards a door. This is a related concept to the icons semantic class "login".

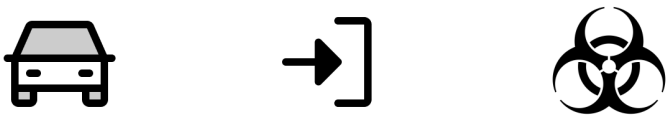

**Figure 4.1:** Icons of three different types. From left to right: representational icon depicting a car, abstract icon containing an arrow pointing towards a door, arbitrary icon containing a biohazard symbol.

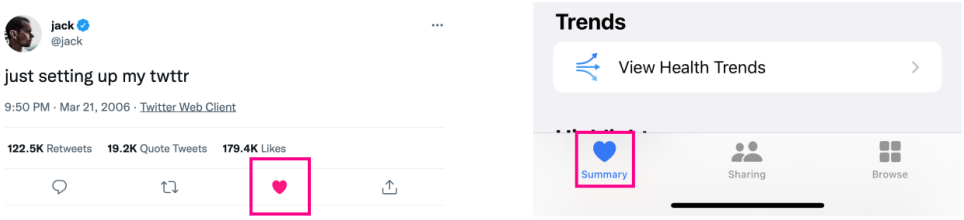


**Figure 4.2:** Screenshot of an icon containing a heart symbol in two different contexts. The heart icon is highlighted in red. Left: Social media page (Twitter `https://twitter.com/Jack/status/20`), Right: Health application (Apple Health `https://www.apple.com/ios/health/`)

How the meaning of an icon is interpreted depends on three key factors [GSF01]:

1. The icon itself
2. The viewer, i. e., the interpretant of the icon
3. The context in which the icon is used

While the icon itself, i. e., its pixel-values, always stays the same, the context and viewer are variable and subjective. Every interpretant of an icon comes with a cultural and social bias that can affect their interpretation [GSF01]. Furthermore, icon semantics may not be obvious for the first time and have to be learned [GSF01]. This is especially true for abstract and arbitrary icons. The context of an item includes factors like the domain of the application the icon is used in or where exactly the icon appears in the graphical user interface.

Figure 4.2 shows an example of how the context of an icon can affect its semantic class. On a social media page, an icon containing a heart symbol has the semantic class “favorite” or “like”. However, the same icon in another context, e. g., a health application, could belong to the semantic class “health” or “heart”.

When labeling icons, we conclude that classifying an icon into a specific semantic class based on solely its pixel values is challenging. It often requires additional knowledge about the context in which the icon is used. This challenge arises both while associating icons with semantic classes during creation of the dataset and while using the final system to classify icons. Related work shows that contextual information in the form of text can be used as additional input to an icon classifier [ZXC21]. For such a multimodal classification approach, contextual information might be text that is located close to the icon [XYX+19; ZXC21] or the internal filename of the icon [XYX+19].

Depending on the underlying platform, such contextual information can be extracted, e. g., from the view hierarchy. As Zang et al.[ZXC21] demonstrate, a multimodel approach can improve model performance compared to an image only approach.

However, this multimodal approach has a key drawback. The final system is not independent of the underlying platform anymore and a screenshot, which is easy to obtain, does not suffice to classify an icon. Different platforms like iOS, Android and the web each have their own representation of the view hierarchy, making it necessary to have platform specific code. This process is vastly more complicated than a screenshot only approach. For example, Xi et al.[XYX+19] resort to using static analysis of Android application packages to extract contextual information for icon buttons. Deep integration with the system is required to obtain metadata at runtime. Furthermore, the icon detection may not run on the device in real time but only on an existing recorded image or video stream, making it impossible to extract non-visual metadata. Therefore, this thesis accepts a possible reduction in semantic accuracy of labels, that can be gained by using additional metadata during classification, in favor of a simpler and more versatile overall system with wider platform and use case support. However, future work may extend on this method by using additional metadata as an optional second input for the classifier. Then the classifier could be used to detect icons based on pixel values only or on pixel values and metadata with improved accuracy.

Sticking to a pixel-only approach requires a trade-off to be made between semantic accuracy and the overall usefulness of the final system. In a pixel-only approach, every icon has to belong to a single semantic class regardless of context. That means, the chosen semantic class is either very broad but correct in the vast majority of contexts, or it is very specific and correct only for a small number of contexts. While the former is less helpful from an accessibility perspective, the latter may be impossible to correctly detect due to the lack of context. An extreme example of a broad semantic class is an icons visual class. This does, however, not mean that all semantic classes are visual classes. Regarding the aforementioned example of an icon containing a heart symbol in the context of a social media and health website, the following two semantic classes could be assigned to the icon: “like” or “heart”. While the semantic class “heart” is less specific and in this case equal to the icon’s visual class, it is correct in the vast majority of contexts. On the other hand, the semantic class “like” is more specific and helpful to the user from an accessibility perspective, but is wrong in the context of a health application.

With regard to the possible use case of the final system being used as support for screen readers as described in Chapter 6, less specific semantic classes require the user to bring contextual information themself. In the example of an icon containing a heart symbol, the user knows the application they use is a social media site and can therefore infer a more specific semantic class themself. Moreover, in a larger system that employs the icon classifier developed in this thesis, an upper layer may use available contextual information unseen by the classifier to map broader semantic classes to more specific semantic classes. For example, if the current context is known to be a social media site, the semantic class “heart” can be mapped to “like” or “favorite”.

We conclude that in the case that an icon clearly belongs to a semantic class in an overwhelming number of contexts, that semantic class should be chosen. For example, an icon depicting a plus symbol virtually always belongs to the semantic class “add”, regardless of context. In other cases where the semantic class of an icon is strongly ambiguous depending on the context, a more broad semantic class should be chosen. In the prior mentioned case of an icon containing a heart symbol,

**Figure 4.3:** Example of compound icons from different icon sets.

we opt to choose the semantic class “heart” instead of “like”. Choosing the correct semantic class during dataset creation can be subjective, and some inconsistencies may be introduced into the dataset.

To obtain a set of objects and actions that are suitable to describe semantic classes of icons in computer programs, the 99 semantic labels by Liu et al.[LCS+18] are chosen to form the basis of the new semantic classes set. These semantic classes were gathered by the authors by analyzing Android applications from a wide variety of categories. Only the most frequently occurring icons were given a semantic class.

After examining the semantic classes, minor adjustments are made. First, some classes, for example “sliders” and “settings”, are merged, as their semantic meaning is close and a differentiation between the two is more of visual than semantic nature. Second, some classes are deleted or renamed, for example “wallpaper” is renamed to the more broad “image” semantic class. Finally, missing classes like “link” or “language”, which are particularly important on the web, are added. A new semantic class “other” is added to store icons which do not belong to any of the other semantic classes. In total, over 113 semantic classes are identified. A list of all semantic classes is shown in Appendix A.

## 4.2 Compound Icons

Another challenge arises when encountering complex icons. Even if the context of an icon is known or assumed, some icons do not fit into a single semantic class consisting of an action or object but require a more complex semantic description. Figure 4.3 shows five icons that have complex semantic classes. The first icon contains an avatar symbol and a plus symbol. Out of the existing semantic classes, the icon could be assigned to either the “add” or “user” semantic class. By choosing either of these classes, as is the case with the approach of Liu et al., information is lost. Ideally, however, the icon would have a semantic class of “add user”.

To further solve this problem, we introduce the terminology of simple icons and compound icons [BSG89; ZKH+20]. Simple icons are icons that depict only a single object or action. Simple icons cannot be further decomposed into other icons, and doing so can result in uninterpretable parts [BSG89]. Compound icons are icons that are made of two or more simple icons and thus can contain multiple actions and objects in their semantics. Examples of compound icons are depicted in Figure 4.3.

Yet, the differentiation between simple and compound icons does not mean that simple icons are limited to a single shape. For example, an icon that contains multiple avatar symbols is not necessarily a compound icon, but a simple icon with the semantic class "group". In some cases, the line between compound and simple icons is blurry.

After observing compound icons and their labels on popular websites (Appendix B) in Germany, including the top 10 ranked pages on SimilarWeb[1], we make two assumptions. First, most compound icons only contain two simple icons and compound icons that contain more than two simple icons are rarely encountered. Second, in the case of compound icons consisting of two simple icons, one of the simple icons typically refers to an action and one refers to an object. The semantic description of the compound icon consequently follows the "action + object" pattern, e. g., "add user" or "delete file". The object icon is commonly the primary icon and is visually dominant, i. e., is larger than the other. Conversely, the action icon is commonly the secondary icon and is visually smaller than the primary icon. It adds additional information to the primary icon. We can observe this pattern in the icons displayed in Figure 4.3. The semantic class of compound icons can be constructed by concatenating the class of the primary icon with the class of the secondary icon. In the aforementioned case of an icon containing an avatar and a plus symbol as depicted in Figure 4.3, the primary icon is the icon with the semantic class "user" and the secondary icon is the icon with the semantic class "add". The semantic class of the compound icon is "add user". All other four icons in Figure 4.3 follow the same pattern.

Based on our observations, we further assume that the majority of compound icons only contain actions regarding the simple creation, modification or deletion (CRUD operations) of data. In particular, the following actions for secondary icons are identified: "add", "edit", "delete", "check" and "search". Figure 4.3 shows an icon of each action type from different open-source icon sets.

We leave it to future work to conduct a study to support or reject our assumptions regarding the usage and properties of compound icons on the web.

We utilize the two assumptions that, first, most compound icons only contain two simple icons and, second, that the secondary icon commonly has the semantics of a CRUD action, to abstract from the complex task of assigning semantic classes to arbitrary compound icons. We split the larger task into two subtasks: First, detecting the primary semantic class of an icon identical to the existing approach by Liu et al. and, second, detecting an additional secondary semantic class of an icon in the form of a CRUD action. To the best of our knowledge, this is a novel approach regarding classification of icons in user interfaces.

In detail, this means, that we restrict the set of icons that can be labeled from all simple and compound icons to only simple icons and those compound icons that contain exactly two simple icons. Compound icons that contain more than two simple icons are ignored. Moreover, compound icons have to include the concept of CRUD action of type "add", "edit", "delete", "check" and "search" within them.

Every icon is then assigned two labels, $y_1$ and $y_2$. $y_1$ is the primary semantic class of the icon, similar to the approach of Liu et al. In the case of a simple icon, $y_1$ is the semantic class of the simple icon. In the case of a compound icon, it is the semantic class of the primary icon, i. e., based

[1] https://www.similarweb.com/top-websites/germany/

| Icon | Primary Label $y_1$ | Secondary Label (CRUD action) $y_2$ | Concatenated Class |
|---|---|---|---|
|  | Time (object) | – | Time |
|  | Check (action) | Check (action) | Check |
|  | User (object) | Add (action) | Add User |
|  | Send (action) | Delete (action) | Delete Send |
|  | Other | Other | – |

**Table 4.1:** Example of primary and secondary semantic classes assigned to icons, including simple and compound icons.

on our assumptions typically the object. The second label $y_2$ tells whether one of the five defined CRUD operators is present in the icon. This may be either in the primary or secondary icon. If no CRUD operator is present, $y_2$ is set to "other" for the icon.

A final list of primary and secondary semantic classes is shown in Appendix A. To better illustrate the labeling of simple and compound icons for the dataset, Table 4.1 shows a list of different icons and their respective primary and secondary labels.

We conclude that this methodology allows us to keep a simple classification approach while still being able to predict useful semantic classes for a large number of compound icons. Related work does not differentiate between compound and simple icons during classification, and therefore is not capable of predicting the same number of semantic classes. Compared to Liu et al. our approach is capable of detecting more than 5 times the number of semantic classes. We increase the number of semantic classes that the final system is capable of outputting from 99 by Liu et al. to $(113 - 5) \cdot 5 = 540$.

## 4.3 Sources

The following icon sources are identified:

1. Existing datasets
2. Open source icon libraries
3. Extracting icons from webpages

### 4.3.1 Existing Datasets

During literature research, the dataset by Liu et al.[LCS+18] is identified as the basis of our new dataset. We refer to this dataset as $D_{\text{liu}}$. $D_{\text{liu}}$ contains over 118663 icons from popular Android applications organized into 99 classes and is based on the Rico dataset [DHF+17]. The authors make their dataset publicly available on the code sharing website GitHub[2]. While the dataset does not contain web icons, the assumption is made that mobile icons are also used on the web. The dataset is split between a taining and validation set.

As all images are cropped from larger screenshots, some images may not be centered perfectly or in rare cases may be missing an icon altogether. However, due to the large number of data points in the dataset, this does not pose a significant problem. Due to the possibility that multiple applications use the same icons, icons in a dataset are not guaranteed to be unique. Images are available in the grayscale color space and the resolution $32 \times 32$ pixels. Icons are upscaled to the required dataset resolution using bilinear interpolation.

As this thesis' semantic classes are based on the semantic classes by $D_{\text{liu}}$, only minor adjustments have to be made to the dataset. While this includes merging and deleting a subset of semantic classes, no further annotation of icons is necessary.

The dataset does not feature a secondary label for icons containing a CRUD action. Due to the overall large number of data points, reannotating all icons with a secondary label is not feasible. However, icons with the primary label "add", "edit", "delete", "check" and "search" are assigned the corresponding secondary label, as they by definition contain a CRUD action. Moreover, icons of the semantic classes "close" and "minus" are assigned to the "delete" secondary class.

### 4.3.2 Open-Source Icon Sets

The dataset is expanded using icons from open-source publicly available icon sets that are designed to be used on the web.

There are two approaches that can be used to obtain these icons. First, whole sets of icons can be manually labeled with semantic classes. Second, icon databases that contain icons from multiple icons sets can be queried for the semantic classes. While the former allows to get an overall more diverse set of icons, the latter allows gathering icons for more rare semantic classes. Both approaches are used in this thesis.

#### Labeling Icon Sets

The following icon sets are identified during online research:

1. Phoshpor Icons[3] (6286 icons)
2. HeroIcons[4] (460 icons)

[2] https://github.com/datadrivendesign/semantic-icon-classifier
[3] https://phosphoricons.com/
[4] https://heroicons.com/

3. Material Icons[5] (10615 icons)
4. Bootstrap Icons[6] (1669 icons)
5. Pixelarticons[7] (460 icons)
6. Skware Icons[8] (100 icons)
7. Feather Icons[9] (287 icons)

For each dataset, all icons are downloaded in the SVG format and subsequently renderer at the target resolution as a PNG image. Icons are assigned a random grayscale foreground and background color, emulating a screenshot. Foreground and background color must however satisfy the WCAG 2.0 success criterion 1.4.3[10] which dictates a contrast ratio between foreground and background of 4.5:1. This ensures that the icon does not visually blend into the background and become undetectable.

In addition to the icon image, if additional metadata is available, it is saved alongside the icon. Metadata might be names, descriptions, keyword, tags or categories that are assigned to the icons by the library authors. Such metadata is often available because it can be used to implement searching or filtering functionality on an icon set's website.

After the icons are downloaded, each icon has to be assigned a primary and secondary semantic label as defined in Section 4.1. To strike a good balance between accuracy and speed of labeling, a semi-automatic approach is used.

First, synonyms for all 113 primary semantic classes and the 5 secondary semantic classes are identified. The open-source Python library NLTK [BKL09], a natural language processing toolkit, is used to search for synonyms. Using the WordNet [Mil95] corpus, a lexical database for the English language, synonyms for words can be identified automatically. Furthermore, additional synonyms are manually added. This is particularly important because some icon sets opt to give their icons filenames that correspond to its visual appearance rather than its semantics. For example, an icon with a paperclip symbol has the filename "paper-clip.png" (HeroIcons). This icon belongs to the semantic class "attachment". "paperclip" however is not automatically identified as a synonym for "attachment" because it is not a synonym in the overall English language but only within the domain of semantic icon classification. Therefore, "paperclip" is manually added as a synonym for "attachment". This process is repeated for other common icon synonyms like "floppy disk" for the semantic class "attachment", "record" for the semantic class "microphone" and "gear" for the semantic class "settings".

Now, a descriptive set of keywords for each icon is generated. This is achieved by concatenating the icon filename with all available metadata about the icon to a descriptive string. Then the descriptive string of the icon is tokenized by splitting it into individual words. During tokenization, a word stemmer included in Python NLTK, namely the snowball stemmer, is used to reduce the word to its

[5] https://fonts.google.com/icons
[6] https://icons.getbootstrap.com/
[7] https://pixelarticons.com/
[8] https://danylpo.com/skware/
[9] https://feathericons.com/
[10] https://www.w3.org/TR/WCAG21/#contrast-minimum

root. For example, the word “bookmarks” is reduced to the root “bookmark”, as is “bookmarked” or “bookmarking”. This allows to match words that have the same root but differ in the way they are conjugated.

Finally, the keywords set of each icon is matched against the 113 semantic classes and their synonyms. This results in a list of possible semantic classes for every icon. In most cases, between 0 and 3 semantic classes are matched. Depending on the size of the set of matched semantic classes, a decision is taken. If the set is empty, the icon is labeled as “other”. If the set contains exactly one semantic class, the icon is labeled as this semantic class. If the list contains more than one class, the icon is marked for further manual annotation.

Manual annotation is performed using a simple custom-built application. The icon is shown to the user and the user can select the correct semantic class out of a list of all available semantic classes. The set of predicted semantic classes is shown on top of all other classes to allow the user to make a quick decision. It is likely that the correct semantic class is an element of the predicted classes.

After the semi-automatic labeling is complete, a quick manual check on the final icon set is performed to detect possible outliers that were wrongly labeled.

The same labeling procedure is repeated for the secondary label, i. e., possible CRUD actions within the icon.

Out of the 7 labeled icon sets, 4 provide multiple stylistic variants of the same icon. These variants are slight stylistic variations of an icon like “regular”, “filled” or “outlined”. The semantic class of an icon stays the same regardless of its variant. Therefore, only one variant set has to be labeled and all other variants of the icons receive the same semantic class. As most icon sets have between 2 and 6 different stylistic variants, this drastically reduces the number of icons to be labeled.

We denote this dataset containing icons from open-source icon sets as $D_{\text{iconsets}}$. $D_{\text{iconsets}}$ is split randomly into two subsets for training and validation. The training set contains 80% while the validation set contains 20% of the icons.

#### Querying Icon Databases

The second approach to gathering icons is to use icon databases. This approach is particularly useful to obtain icons for semantic classes that are less common and thus appear more seldom in the icon sets labeled in Section 4.3.2.

Icon databases are databases that contain icons from a multitude of origins. Based on related work[LGG19], the Noun Project[11] database is used. The Noun Project contains over 3 million indexed icons from various sources. The database is available as an online service and provides an HTTP API to interface with it. The Noun Project is queried for each semantic class and the first 200 results are retrieved. Icon images are not offered in the SVG format for noncommercial customers, but a high-res transparent PNG image is available. Icons resized and assigned a foreground and background color in the same process as stated in the previous section.

The process is repeated for secondary labels.

[11] https://thenounproject.com/icons/

In total over 24795 icons are collected. We denote this dataset as $D_{\text{database}}$. Again, the dataset is split randomly into two subsets for training(80%) and validation(20%) .

### 4.3.3 Scraping Icons from Websites

The final set of icons are extracted from 19 popular real-world webpages. This allows us to add icons from non publicly available sources. Websites include the top-ranked pages on SimilarWeb in Germany. A full list of the websites can be found in Appendix C.

Icons are extracted from the websites using a custom-built application. The TypeScript programming language is used for the implementation. This allows for code-sharing between the browser extension built in chapter Chapter 6 and the icon extractor. We utilize Puppeteer[12], an open-source JavaScript library that provides a high-level API to control a Chromium-based browser. The program autonomously navigates to the defined websites and identifies all icons using heuristics further described in Chapter 6. After all possible icons are identified, the program takes a high-resolution screenshot of each icon and saves it to disk. If additional metadata, like aria-label tags, is available, it is saved alongside the icon. Detected elements that are not icons, i. e., false positives, are manually identified and removed.

All icons are manually annotated with a primary and secondary label. Manual labeling is preferred over the semi-automatic labeling approach outlined in Section 4.3.2 because the collected icons are used to evaluate the classifiers developed in Chapter 5 on a real-world task. To gain the most accurate evaluation results, a correct ground truth is required.

Using this method, a total of over 412 icons are collected. We denote this dataset by $D_{\text{web}}$. We reserve these icons in order to evaluate the trained classifiers on a real-world task. Hence, these icons are not added to the overall dataset.

We define the final dataset $D_{\text{all}}$ as

$$D_{\text{all}} = D_{\text{mobile}} \cup D_{\text{iconsets}} \cup D_{\text{database}} \tag{4.1}$$

A final overview of the collected dataset is given in Table 4.2.

| Descriptor | Icons |
|---|---|
| $D_{\text{liu}}$ | Icons from mobile Android applications organized into 99 semantic classes by Liu et al. |
| $D_{\text{mobile}}$ | $D_{\text{liu}}$ with updated 113 semantic classes. |
| $D_{\text{iconsets}}$ | Icons collected from various open-source icon libraries |
| $D_{\text{database}}$ | Icons collected from icon databases. |
| $D_{\text{all}}$ | $D_{\text{mobile}} \cup D_{\text{iconsets}} \cup D_{\text{database}}$ |
| $D_{\text{web}}$ | Icons collected from popular real-world websites. $D_{\text{web}}$ is reserved for validation ($D_{\text{web}} \notin D_{\text{all}}$). |

**Table 4.2:** Overview of different collected datasets.

[12] https://pptr.dev/

## 4.4 Data Analysis

Before a classifier is trained on the new dataset, it is analyzed to gain further insights into the collected icons.

The differences in style and appearance between the different icon sets are examined. We expect to find that icons on the web are more diverse, i. e., by average less similar in style to each other, than on mobile platforms for reasons outlined in the introduction of this theis. Furthermore, we expect that icons from a single icon set that are designed and published together, e. g., icon libraries like HeroIcons, Phoshpor Icons or Material Icons, share a common style and visual identity.

Based on our expectations, we set up the following four hypotheses:

1. Icons on the web are less similar to another in style than they are on mobile platforms.
2. Icons on the web are less similar to another in style than they are in a single icon set.
3. Icons within a single icon set are similar in style.
4. The similarity is higher for icons within a single icon set than it is for icons between two icon sets.

To analyze the differences in style between icons, we employ a deep learning approach published by Lagunas et al.[LGG19]. Lagunas et al. propose a similarity metric that aims to capture differences in style and visual identity of icons. Compared to other methods for measuring style similarity, the authors claim their approach focuses not only on low-level stylistic features such as stroke width or symbol curvature, but on capturing the overall visual identity of an icon. Lagunas et al. train a siamese neural network to differentiate between similar and dissimilar icons. The resulting network can be used to map icons into a high dimensional feature space that represents the icon's overall style. Using the feature vectors of two icons, the euclidean distance is used to measure the distance between them. A distance of 0 indicates that the icons are identical in style. The greater the distance, the less similar two icons are. Lagunas et al. make their code and weights for the neural network publicly available[13].

Figure 4.4 shows a clustering by Lagunas et al. of icons based on the similarity metric. Icons that are similar in style are grouped together. For example, icons that feature a circular filled style are grouped in the top right of the visualization, while thin square icons are located in the bottom right. Only the style similarity and visual identity of an icon is relevant, as is shown in the different highlighted clusters by Lagunas et al. Clusters may contain icons of different semantic classes.

The examined icon sets include icon sets labeled in Section 4.3.2 as well as $D_{\text{mobile}}$ and $D_{\text{web}}$. For icon sets that are published in multiple stylistic variations, e. g., "outlined" or "filled", we choose the default variation of the icon set. In detail, we compare HeroIcons (outline), Phosphor Icons (regular), Material Icons (outline), Pixelarticons, Skware, $D_{\text{mobile}}$ and $D_{\text{web}}$

We note that we cannot prove that $D_{\text{mobile}}$ and $D_{\text{web}}$ are suitable as input for the neural network published by Lagunas et al. The network expects a high resolution icon image with $180 \times 180$ pixels as input. $D_{\text{mobile}}$ however only features icons with $32 \times 32$ pixels in resolution that have to be

[13] https://github.com/mlagunas/learning-icons-appearance-similarity

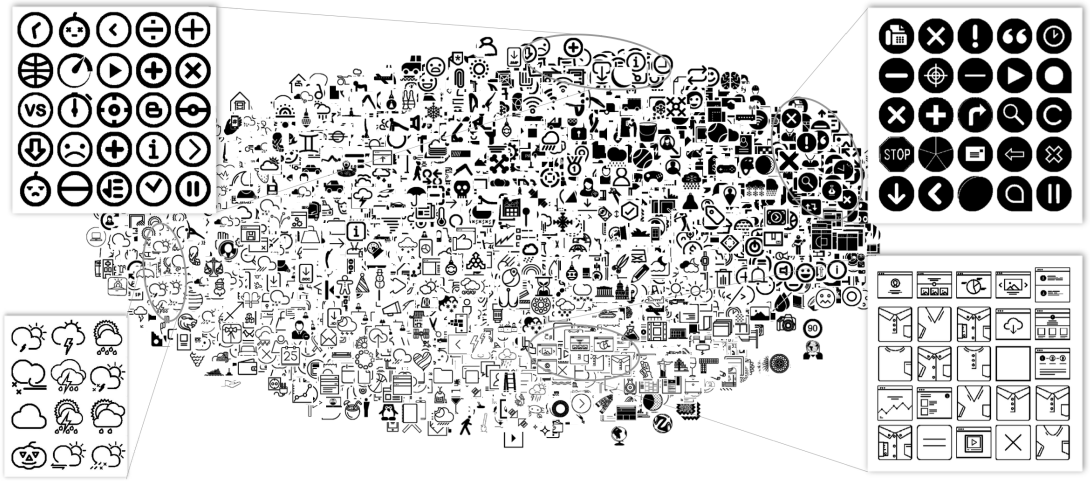

**Figure 4.4:** Clustering visualization of the simliarity metric by Lagunas et al.[LGG19]. Lagunas et al. use the t-SNE algorithm[VH08] to reduce the dimensionality of icon features vectors to a two-dimensional space. Icons that are similar in style and appearance are grouped together. Illustration by Lagunas et al.[LGG19]

.

upscaled. Furthermore, example images shown in the publication of Lagunas et al. only feature black icons on a white background. This contradicts the two dataset $D_{\text{mobile}}$ and $D_{\text{web}}$ which both contain an arbitrary grayscale foreground and background color. Moreover, we cannot prove that the outputs of the neural network match with results obtained by Lagunas et al., as no test data with known results is publicly available. Python code and network weights are taken from the research papers[LGG19] GitHub repository without modifications. Nevertheless, we report the results in the coming sections under the assumption that these icon images are suitable as input to the neural network.

To reject or accept the prior stated hypotheses, we use the similarity metric proposed by Lagunas et al. to compare the style of icons in two ways. First, the similarity in style within different icon sets is compared. Second, the style similarity between two icon sets is compared.

To compare the similarity of icons within a given icon set, the feature vectors for each icon are computed using the neural network proposed by Lagunas et al.[LGG19]. Then the pairwise euclidean distances between each of the icons feature vectors are calculated. The mean, median, standard deviation and quartiles of the pairwise euclidean distances are calculated. Figure 4.5 shows the results as a box plot. The mean and standard deviation are reported in Table 4.3.

We find that the mean similarity of all icon sets falls between 0.25 and 0.4. We do not find that icons within a icon set of single origin set are more similar in style by average than icon sets which contain icons from multiple origins, i. e., $D_{\text{mobile}}$ and $D_{\text{web}}$. Both $D_{\text{mobile}}$ and $D_{\text{web}}$ feature a mean and median pairwise similarity that is comparable to that of Phosphor icons or Material icons. This does not support our second hypothesis. However, we note that the distributions of the pairwise similarities in $D_{\text{mobile}}$ and $D_{\text{web}}$ are flatter than that of icon sets with a single origin. With 0.155964 for $D_{\text{web}}$ and 0.195923 for $D_{\text{mobile}}$, the standard deviation is higher than for any icon sets of a single origin. This fact is also represented by the large center quartiles for $D_{\text{mobile}}$ and $D_{\text{web}}$ in the box

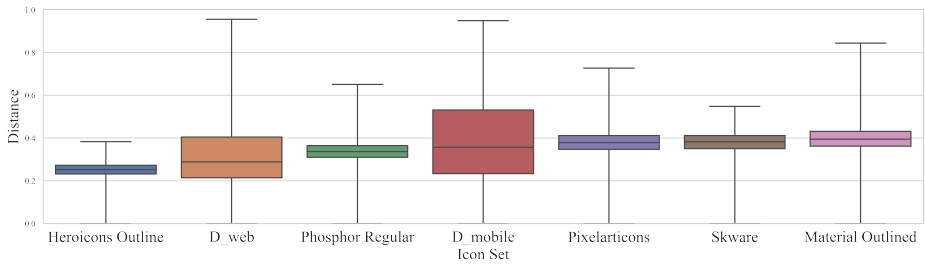


**Figure 4.5:** Pairwise icon similarity within different icon sets, visualized as a box plot. The box shows the middle two quartiles. Whiskers show the upper and lower quartiles. Icon sets are sorted by median in ascending order.

| Set | Mean Pairwise Similarity | Pairwise Simliarity Standard Deviation |
|---|---|---|
| Heroicons | 0.250 | 0.039 |
| D_web | 0.323 | 0.156 |
| Phosphor | 0.337 | 0.046 |
| Skware | 0.375 | 0.067 |
| Pixelarticons | 0.379 | 0.059 |
| D_mobile | 0.390 | 0.196 |
| Material | 0.399 | 0.063 |

**Table 4.3:** Mean and standard deviation of the parwise style similarity between all icons in a given icon set. Rows are sorted by mean pairwise similarity in ascending order.

plot shown in Figure 4.5. This means that there is more variance in style similarity on the web and mobile icons than in icon sets with a single origin. This is also reflected in the fact that the maxima of the pairwise similarities of $D_{\text{mobile}}$ and $D_{\text{web}}$ are over 12.5% higher than for any icon sets of a single origin. There are icon combinations that are less similar in style than any combination of icons from icon sets of a single origin.

We find that HeroIcons mean similarity is 25% lower than that of the next similar icon set. HeroIcons also features the lowest standard deviation across all icon sets with 0.039. These findings indicate that HeroIcons has the most coherent visual appearance of the examined datasets.

The data does not support the first hypothesis that icons on the web are more diverse in style and appearance than on mobile platforms. We find that with 0.390 $D_{\text{mobile}}$ features a 20.5% higher mean pairwise similarity than $D_{\text{web}}$ with 0.323 indicating the opposite of the expected behavior.

As the similarity metric cannot be interpreted as an absolute value, we cannot draw any conclusions with respect to hypothesis three.

In the next step, we examine the style similarity between different icon sets. This means, the distances are not computed pairwise within the same set of icons, but pairwise between two sets. The similarity metric is applied to measure the distance between every icon in one set with every icon in the other set. Based on hypothesis 4 we expect the differences in style to be higher between

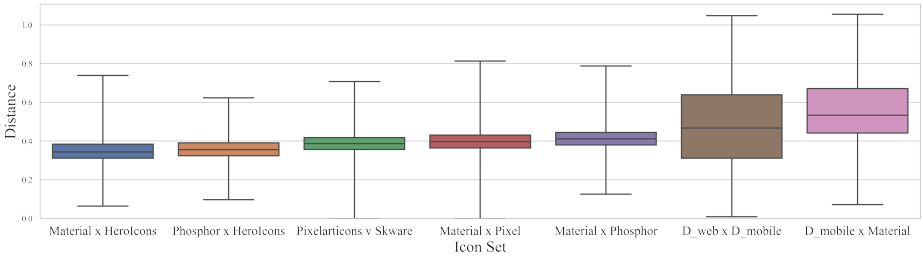


**Figure 4.6:** Pairwise icon similarity between different icon sets, visualized as a box plot. The box shows the middle two quartiles. Whiskers show the upper and lower quartiles. Icon sets are sorted by median in ascending order.

| Set 1 | Set 2 | Mean Pairwise Similarity | Pairwise Simliarity Standard Deviation |
|---|---|---|---|
| Material Icons | HeroIcons | 0.352 | 0.060 |
| Phosphor Icons | HeroIcons | 0.359 | 0.052 |
| Pixelarticons | Skware | 0.388 | 0.049 |
| Material | Pixelarticons | 0.401 | 0.059 |
| Material Icons | Phosphor Icons | 0.414 | 0.055 |
| D_web | D_mobile | 0.473 | 0.203 |
| D_mobile | Material | 0.553 | 0.145 |

**Table 4.4:** Mean and standard deviation of the parwise style similarity between icons in set 1 and set 2. Rows are sorted by mean pairwise similarity in ascending order.

two icon sets than within the two. Due to the large amount of possible combinations, only a subset of the combinations are reported. Figure 4.6 shows the results as a box plot. The mean and standard deviation are reported in Table 4.4.

The results show that the mean pairwise similarity between different icon sets falls into a slightly higher range than the similarity within different icon sets. All mean pairwise similarities between icon sets of a single origin are in the range $[0.35, 0.41]$. While the upper boundary is comparable to that of within icons, the lower boundary is 40% higher. This does support the second hypothesis that icons are more similar within themselves than between them. However, when we exclude HeroIcons from the comparison the range of mean similarities is much closer with a range of $[0.38, 0.41]$ between sets and $[0.34, 0.4]$ within sets.

We find that the during the style comparison between $D_{\text{web}}$ and $D_{\text{mobile}}$ the pairwise similarity features a higher mean and standard deviation than any comparison between single origin icon sets with a mean of 0.47 and standard deviation of 0.2. Again, this comes expected as to the larger variety of icons from multiple origins.

We conclude that analyzing icon sets with the similarity metric proposed by Lagunas et al. does not support any of our three hypothesis with sufficient confidence. We acknowledge possible limitations of this approach regarding the dataset $D_{\text{mobile}}$ and $D_{\text{web}}$ as outlined previously.

# 5 Classifier

In this chapter, we train a classifier on the dataset and evaluate its performance. We define the following functional requirements for the classifier:

1. It takes the pixel values of an icon as input.
2. It outputs a semantic class.

Based on the success of related work[LCS+18], we choose to use a convolutional neural network as the classifier.

## 5.1 Model Architecture

With the use case of improving accessibility on the web in mind, we identify the following requirements for the model architecture:

1. It provides state-of-the-art performance.
2. It provides fast inference times ($< 1s$) even on resource constrained devices .

Based on these requirements, the MobileNetV2 architecture is chosen. More specifically, MobileNetV2 is used as a feature extractor. Related work demonstrates that this architecture is capable of achieving high accuracy on the task of icon classification[BS; FMY+21].

MobileNetV2 is an advanced CNN architecture proposed by Sandler et al. in 2018 [SHZ+18]. It is part of the MobileNet[HZC+17] family of networks and provides state-of-the-art performance for mobile image classification. MobileNetV2 aims to achieve fast and memory efficient inference on mobile devices by reducing its computational costs. These properties also make it suitable for other resource constrained hardware and environments like webpages and browser extensions. MobileNetV2 achieves this by employing multiple techniques, including the concept of depthwise separable convolutions.

As Sandler et al. state, depthwise separable convolutions are a core building block for many convolutional neural network architectures that focus on performance and efficiency[HZC+17; SHZ+18]. Compared to regular convolutional layers, depthwise separable convolution splits up the convolution operation into two steps: a depthwise convolution and a pointwise convolution. During depthwise convolution, the input image gets convolved with the filter independently channel by channel. This is different from regular convolution, where all channels are convolved with the filter at the same time. This means the depthwise convolution filter has a depth of one and does not extend throughout the whole image depth. The result is an activation map for each channel of the input image. All channel activation maps stacked form a tensor with the same depth as the

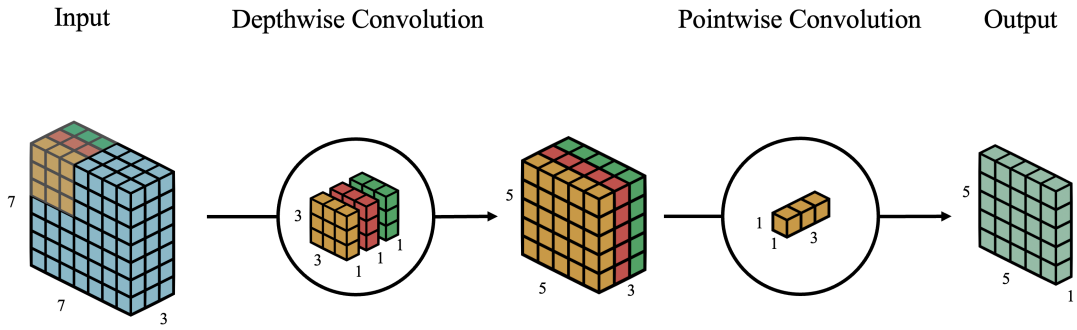


**Figure 5.1:** Illustration of a depthwise separable convolution consisting of a depthwise convolution and a pointwise convolution.

input. The output of the depthwise convolution is then passed to the pointwise convolution. During pointwise convolution, a 1x1 filter with full depth is applied to the input. The result is a tensor that has the same shape that a normal convolution would have produced.[SHZ+18].

Figure 5.1 shows an example of a depthwise separable convolution of an image tensor of shape $7 \times 7$ with three channels. First, the input tensor, is convolved depthwise, i. e., channel by channel, with three different filters. The resulting output tensor has a depth of 3. The output is then convolved pointwise with a $1 \times 1$ filter with full-depth. The output is a two-dimensional tensor that has the same shape as the output of a regular convolution.

While depthwise separable convolutions first appear to be less powerful than regular convolutions, according to empirical evidence, CNNs that use depthwise separable convolutions perform similarly to standard convolutional layers [SHZ+18]. The main advantage of depthwise separable convolution, however, comes in the form of a reduction of the computational cost. Depthwise separable convolutions can greatly reduce the number of parameters and number of computations necessary. While a regular convolution layer has a computational cost of $h \cdot w \cdot d_i \cdot d_j \cdot k \cdot k$, depthwise separable convolution layers only have a cost of $h \cdot w \cdot d_i \cdot (k \cdot k + d_j)$, where $h$ is the input tensor height, $w$ the input tensor width, $d_i$ the input tensor depth, $d_j$ the number of filters and $k$ the filter size [SHZ+18]. This is a reduction by a factor of $\frac{k^2 d_j}{k^2 + d_j}$ [SHZ+18]. In the case of MobileNetV2, which utilizes 3x3 convolutions, this means an 8-9 times reduction in the number of computation necessary.

The concept of depthwise separable convolution together with inverted residuals with linear bottlenecks [SHZ+18] form the primary building block for MobileNetV2. MobileNetV2 contains a total of 19 of these bottleneck blocks. Additionally, batch normalization and dropout are utilized, which can help to make the network learn faster and prevent overfitting[GWK+18].

The output of MobileNetV2 is a tensor of shape $(3, 3, 1280)$ that we downsample to a one-dimensional feature vector of the length 1280 by applying adding an average pooling layer. A final dropout and fully connected layer with the number of neurons equating to the number of semantic classes is added on top of the MobileNetV2 feature extractor to make the model ready to be used for

| Layer Name | Output Shape | Number of Parameters |
|---|---|---|
| Input Layer | (96, 96, 3) | – |
| MobileNetV2 | (3, 3, 1280) | 2257984 |
| GlobalAveragePooling2D | (1280) | – |
| Dropout | (1280) | – |
| Dense | (113) | 144753 |

**Table 5.1:** Layer stack of the MobileNetV2 based model.

| Layer Name | Output Shape | Number of Parameters |
|---|---|---|
| Input Layer | (96, 96, 3) | – |
| Conv2D | (96, 96, 16) | 448 |
| MaxPooling2D | (48, 48, 16) | – |
| Conv2D | (48, 48, 32) | 4640 |
| MaxPooling2D | (24, 24, 32) | – |
| Conv2D | (24, 24, 64) | 18496 |
| MaxPooling2D | (12, 12, 64) | – |
| Dropout | (12, 12, 64) | – |
| Flatten | (9216) | – |
| Dense | (128) | 1179776 |
| Dense | (113) | 14577 |

**Table 5.2:** Layer stack of the baseline CNN.

classification. The output of the network is a one-hot encoded vector of the length of the number of classes. The value 1 on position $i$ indicates that the icon is of class $i$. All other entries are set to zero. Table 5.1 shows the final stack of layers for the MobileNetV2 network.

To establish a baseline, a straightforward convolutional neural network is also trained on the same datasets. We base the architecture of the baseline model on the TensorFlow guide to image Classification[1]. The network consists of three convolutional blocks, with one convolutional and pooling layer each. After a flattening operation, two fully connected layers follow. Just like the MobileNetV2 based model, a dropout layer is used before the final fully connected layers. An overview of the layers of the baseline CNN is shown in figure Table 5.2.

## 5.2 Data Preprocessing and Data Augmentation

During the training process, data preprocessing and data augmentation are performed.

Icon images are preprocessed by converting them to grayscale and scaling all pixel values from the interval $[0, 255]$ to $[-1, 1]$. Scaling in $[-1, 1]$ instead of $[0, 1]$ or $[0, 255]$ is required by MobileNetV2[SHZ+18]. Data preprocessing is performed by prepending a rescaling layer to the

[1] https://www.tensorflow.org/tutorials/images/classification

model. This layer rescales the input tensors values dynamically during the training process without having to statically alter all training data. The same rescaling layer is applied to the baseline model to ensure comparability between the two.

As an additional preprocessing step, ZCA whitening has been used by related work [LCS+18]. However, ZCA whitening is explicitly not used in this work, as it restricts interoperability of the model with the browser extension developed in Chapter 6.

In addition to data preprocessing, data augmentation is performed. Data augmentation describes the process of creating more training data from the original dataset by randomly applying slight mutation like flipping or scaling to every input image [Ten22; WSD+17]. This is important because CNNs with many hidden layers like MobileNetV2 require a large amount of training data[WSD+17]. Data augmentation is implemented by prepending data augmentation layers to the preprocessing layers. Data augmentation layers are disabled after the network has finished training, i. e., they are not enabled during inference and only pass through data. The following data augmentation techniques are performed:

1. Contrast ratio adjustment by a random factor in the range $[-0.1, 0.1]$.
2. Translation on the x and y-axis by a random factor in the range $[-0.1, 0.1]$ of the overall image width and height.
3. Zoom in by a random factor in the range $[0, 0.05]$.

Randomness for all data augmentation techniques is distributed uniformly.

Other data augmentation techniques like random rotation are not applied, as they could alter the semantic class of an icon. For example, an arrow pointing to right could be randomly rotated by 45 deg to point to top right, which would change its semantic class from “arrow_r” to “arrow_tr”. This introduces unwanted errors into the dataset. The same argumentation applies to other data augmentation techniques, like random flipping.

An example of data augmentation is shown in Figure 5.2.

## 5.3 Implementation

The neural networks are implemented using TensorFlow[2][MAP+15]. TensorFlow is an open-source machine learning framework developed by Google. TensorFlow is designed to efficiently implement and run machine learning algorithms, including deep neural networks [MAP+15]. It has been used successfully in a variety of different fields including icon classification and object detection[LCS+18]. The high-level TensorFlow Keras API[3] is used for its simplicity and flexibility. TensorFlow is chosen over the competitive PyTorch framework[PGM+19] because it offers a wide range of client libraries across different platforms and programming languages. In particular, Tensorflow does not

[2] https://www.tensorflow.org/

[3] https://www.tensorflow.org/api_docs/python/tf/keras

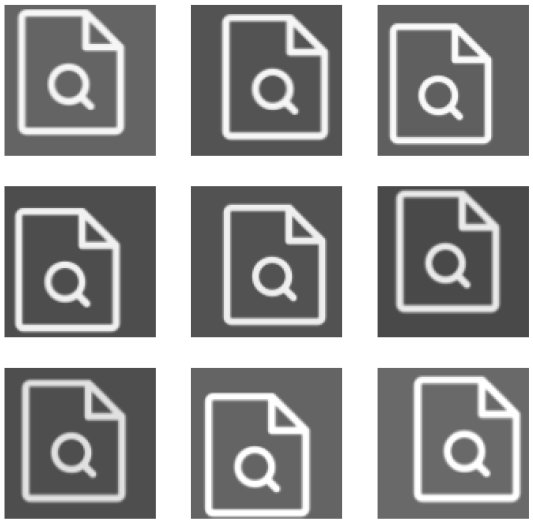

**Figure 5.2:** Grid of 9 examples of data augmentation of a single icon image with applied random contrast, brightness, translation and zoom.

only publish a Python client library but also a Tensorflow.js[4], which is a version of TensorFlow that can run in JavaScript environments like Node.JS[5] or web browsers. This is required for the implementation of the browser extension.

MobileNetV2 is available as a ready to use architecture in TensorFlow and required no custom implementation.

## 5.4 Apparatus

The following apparatus is used to train the two neural networks:

1. University of Stuttgart Institute for Parallel and Distributed Systems Analytic Computing machine learning server
2. Google Colab[6]

[4] https://www.tensorflow.org/js
[5] https://nodejs.org/en/
[6] https://colab.research.google.com/

## 5.5 Training

Both the baseline model and MobileNetV2 are trained on the datasets $D_{\text{liu}}$, $D_{\text{mobile}}$ and $D_{\text{all}}$ (Overview Table 4.2). We differentiate between the primary and secondary labels of the datasets. The notation $D_{\text{X, primary}}$ and $D_{\text{X, secondary}}$ is used to refer to the dataset $D_{\text{X}}$ with only primary or secondary labels where $D_{\text{X}}$ is either $D_{\text{liu}}$, $D_{\text{mobile}}$ or $D_{\text{all}}$.

To train the MobileNetV2 based model, transfer learning is used. We base our approach and code on the TensorFlow guide to transfer learning and fine-tuning for image classification[Ten22]. Transfer learning is based on the premise that if a model is trained on a sizable enough dataset with appropriate generality, it learns to extract features from its input data that are also useful for similar tasks [HAE16; Ten22]. Transfer learning allows for faster training times because the model can re-use weights already learned previously instead of starting from scratch. Moreover, a possibly more accurate final model can be obtained, due to the model having seen a larger overall dataset[Ten22]. In concrete terms, this means that the network will not start with randomly initialized weights, but with weights that were obtained by training the model on a similar dataset. Various pre-trained network weights for MobileNetV2 are made available by the TensorFlow developers. The chosen pre-trained network was trained on the ImageNet dataset [DDS+09]. ImageNet is a large publicly available dataset designed to train neural networks for image classification [DDS+09]. Transfer learning based on the ImageNet dataset has been employed successfully in a variety of different fields [HAE16].

Transfer learning is applied in two steps[Ten22]. In a first step, the MobileNetV2 feature extractor is loaded with the pre-trained weights. The loaded weights are then immediately frozen, i. e., fixed during training. This means that the only weights and biases in the final fully connected layer can be adjusted during the training process. This behavior is chosen so that preloaded weights in the feature extractor are not immediately overwritten. The assumption of transfer learning is that these weights are already generically useful[Ten22]. After that, a fine-tuning step is performed. During fine-tuning, the feature extractor is unfrozen and parameters throughout the whole network are trained. It is common that during fine-tuning, the learning rate of the network is decreased [Ten22].

The baseline network is trained without transfer learning, as no pre-trained weights are available for the custom architecture.

As a loss function for the network, the categorical cross entropy loss function is utilized. The categorical cross entropy loss function is a loss function that is commonly used for classification tasks and measure the difference between two one-hot encoded vectors.

The Adam optimizer[KB15] based on the stochastic gradient descent method is used. According to empirical findings, Adam outperforms other stochastic optimization techniques [KB15]. We use mini batches of size 32 and train the network for 20 epochs pre fine-tuning and 20 epochs post fine-tuning. The baseline model is trained for 40 epochs.

Figure 5.3 shows the loss and accuracy of the MobileNetV2 model during the training process of $D_{\text{all, primary}}$. While the model already performs well after the initial training step with fixed feature extractor weights, fine-tuning the model brings a noticeable improvement in accuracy and reduction in loss. We observe that the training loss and accuracy is smaller than the validation loss and accuracy, indicating slight overfitting on the dataset.

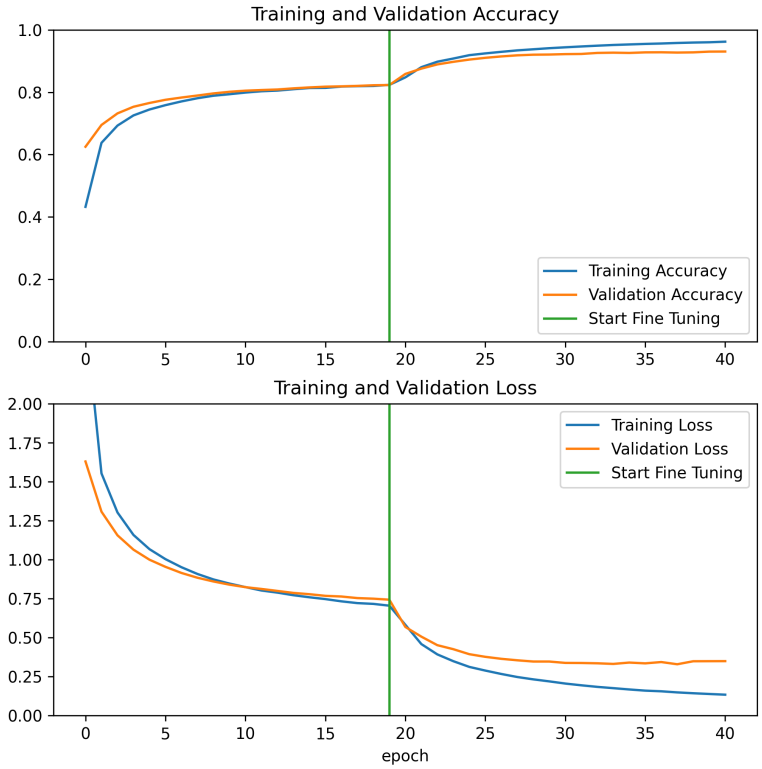


**Figure 5.3:** Loss and accuracy of the MobileNetV2 model during training on $D_{\text{all, primary}}$.

## 5.6 Evaluation

The following metrics and techniques are used to evaluate the model:

1. Accuracy
2. Precision, recall and F1 score for each semantic class
3. Confusion matrix
4. Inference time

### 5.6.1 Accuracy

In this section, we compare the training and validation accuracy of the networks. We focus on finding differences between the different datasets collected in Chapter 4.

We compare the training and validation accuracy of MobileNetV2 and the baseline CNN on the two datasets $D_{\text{mobile}}$ and $D_{\text{all}}$. Training and validation splits of the datasets are as defined in Chapter 4. Furthermore, we compare the accuracy of the models on the dataset $D_{\text{liu}}$ to establish a comparison

| | Baseline CNN | MobileNetV2 | Liu et al.[LCS+18] |
|---|---|---|---|
| $D_{\text{liu}}$ | 0.9273 / 0.9137 | 0.9871 / 0.9464 | 0.9949 / 0.9443 |
| $D_{\text{mobile, primary}}$ | 0.9310 / 0.9116 | 0.9875 / 0.9442 | – |
| $D_{\text{mobile, secondary}}$ | 0.9846 / 0.9895 | 0.9990 / 0.9962 | – |
| $D_{\text{all, primary}}$ | 0.8807 / 0.8820 | 0.9624 / 0.9307 | – |
| $D_{\text{all, secondary}}$ | 0.9721 / 0.9677 | 0.9933 / 0.9729 | – |

**Table 5.3:** Model accuracy evaluation matrix (train/ validation).

| | Accuracy on $D_{\text{web, primary}}$ |
|---|---|
| Trained on $D_{\text{mobile, primary}}$ | 0.83 |
| Trained on $D_{\text{all, primary}}$ | 0.89 |

**Table 5.4:** Model Accuracy on $D_{\text{web, primary}}$ after being trained on $D_{\text{mobile, primary}}$ and $D_{\text{all, primary}}$.

against the CNN developed by Liu et al.[LCS+18]. Training and validation accuracy for the CNN developed by Liu et al.[LCS+18] are taken from the publication "Learning Design Semantics for Mobile Apps"[LCS+18] and the corresponding GitHub repository[7]. We differentiate between the primary and secondary labels in each dataset. The resulting evaluation matrix is shown in Table 5.3.

We find that MobileNetV2 performs equally well as the CNN proposed by Liu et al. on the dataset $D_{\text{liu}}$. While the training accuracy of MobileNetV2 is marginally lower with 0.9871 by less than one percent with, the validation accuracy is equal to the CNN by Liu et al. MobileNetV2 achieves an overall validation accuracy of 98.71% on the dataset. These results underline the suitability of the MobileNetV2 CNN for the task of icon semantic classification. The baseline CNN performs worse than either of the other two networks, with about a 6% decrease in training and a 3.3% decrease in validation accuracy compared to MobileNetV2.

The baseline CNN and MobileNetV2 both perform as well on the dataset $D_{\text{mobile, primary}}$ as they do on $D_{\text{liu}}$. As $D_{\text{mobile, primary}}$ contains the same icons as $D_{\text{liu}}$ but with updated semantic classes, we conclude that merging or deleting classes of $D_{\text{liu}}$ does neither worsen nor improve the accuracy of the networks.

We find that the overall training and validation accuracy decreases on the dataset $D_{\text{all}}$ compared to $D_{\text{mobile}}$. This is true for both MobileNetV2 and the baseline CNN. When predicting the primary labels, MobileNetV2 looses 2.5% in training and 1.4% in validation accuracy on $D_{\text{all}}$ compared to

| | Accuracy on $D_{\text{web, secondary}}$ |
|---|---|
| Trained on $D_{\text{mobile, secondary}}$ | 0.93 |
| Trained on $D_{\text{all, secondary}}$ | 0.97 |

**Table 5.5:** Model Accuracy on $D_{\text{web, secondary}}$ after being trained on $D_{\text{mobile, secondary}}$ and $D_{\text{all, secondary}}$.

[7] https://github.com/datadrivendesign/semantic-icon-classifier

$D_{\text{mobile}}$. The baseline CNN losses more than 5% in training and 3% in validation accuracy. These numbers indicate that the dataset $D_{\text{all}}$ is overall more complex than $D_{\text{mobile}}$. The reduction is less pronounced when predicting secondary labels for both network architectures.

Finally, we examine the differences in MobileNetV2 accuracy on the dataset $D_{\text{web}}$ after the networks have been trained on $D_{\text{mobile}}$ and $D_{\text{all}}$. This is different from previous comparisons, as accuracy is not reported on the same dataset the models are trained on. $D_{\text{web}}$ contains icons from real-world websites that are neither part of $D_{\text{mobile}}$ nor $D_{\text{all}}$ and are therefore unseen to the networks. This comparison therefore demonstrates the performance of the models on a real-world task, as it would be encountered, e. g., in the browser extension developed in Chapter 6.

Table 5.4 shows the resulting accuracy for primary labels. We find that overall the accuracy is lower on $D_{\text{web}}$ compared to the accuracy of the two training set $D_{\text{mobile}}$ and $D_{\text{all}}$. However, the validation accuracy increases substantially by over 6% after the networks have been trained on $D_{\text{all}}$ compared to $D_{\text{mobile}}$. This supports our hypothesis that icon classification on the web is a can benefit from a more diverse dataset than just mobile icons.

We observe a similar effect for secondary icons where the accuracy increases from 0.93 to 0.97 when additional icons are added to the training set as shown in Table 5.5.

Based on the accuracy evaluation, we conclude that MobileNetV2 is a capable CNN architecture for classifying icon images and performs equally well than the CNN proposed by Liu et al. while outperforming the baseline model. The data confirms our hypothesis that adding additional icons to the mobile icons only dataset can improve classifier performance on icons found on real-world websites.

### 5.6.2 Precision, Recall, F1 score and Confusion Matrix

We compute the precision, recall and F1-score for each primary and secondary semantic class for the network MobileNetV2 and the final dataset $D_{\text{all}}$. The full list of precision, recall and F1 scores is shown in the appendix Appendix D. Furthermore, the confusion matrix is computed and displayed in Figure D.1.

While most precision, recall and F1 scores are high for all primary and secondary semantic classes, we find that that precision of the primary semantic class “other” is low with a score of 0.316. The semantic class “other” contains all icons that do not belong to any other class. The low score indicates that only a third of all icons that are predicted to be of the class “other” are actually icons of the class “other”. However, the recall score of the “other” class is substantially higher with 0.820. This equates to 82% of icons that belong to the class “other” are detected as such.

With a precision of 0.962 and recall of 0.997 we do not find the same behavior for the secondary semantic classes “other” which can be traced back to reduced number of classes from 113 to 6.

Other primary semantic classes with a low precision ($\leq 0.5$) include “arrow_tr” (arrow pointing to the top right), “arrow_bl” (arrow pointing to the bottom left) and “arrow_br” (arrow pointing to the bottom right). Primary semantic classes with a low recall score ($\leq 0.5$) include “zoom_in”, “arrow_br” and “chart”

| | Single Sample | Batch of 32 |
|---|---|---|
| Baseline | 17.80ms | 26.66ms |
| MobileNetV2 | 22.54ms | 56.19ms |

**Table 5.6:** Inference Time for baseline CNN and MobileNetV2 on a single sample and a batch of 32.

While examining the confusion matrix, we find that classes with semantic overlap like “close” and “delete” are more often confused than classes with no semantic overlap. For example, depending on the context, an icon depicting a minus symbol could be assigned to either of the two classes. Other example for this behaviour include “compare” and “swap” or “list” and “playlist”.

### 5.6.3 Inference time

Evaluating inference time is particularly important with respect to the requirement Item 2 and browser extension developed in Chapter 6. A short inference time is crucial for the usability of the browser extension, especially on resource constrained devices. Measuring inference time is performed by averaging over 100 runs performed using TensorFlow version 2.8.0 with GPU support disabled on an Apple M1 Pro CPU. Table 5.6 shows the inference time of the baseline CNN and MobileNetV2 model on a single sample and a batch of 32 images.

We find that inference time for both models is low and satisfy the requirement Item 2. While MobileNetV2 takes 4.7ms longer than the baseline model to predict a single sample, the absolute value is still fast with 22.54ms compared to 17.8ms for the baseline model. Surprisingly, we do not find a large difference in inference time between a batch of 32 images and a single sample. Predicting a semantic label for 32 samples at a time takes just 33.65ms longer than predicting a single sample for MobileNetV2. A similar difference is observed for the baseline model.

# 6 Application

In this chapter, we develop a browser extension that automatically attaches missing accessibility metadata to icons on websites using the classifier developed in the previous chapter.

## 6.1 Requirements

The following functional requirements for a browser extension are identified:

1. Icons elements on the website are automatically labeled using the classifier developed in Chapter 5 and accessibility metadata is attached to the icon.
2. The user can trigger icon labeling using a control pane.
3. The user can trigger icon labeling using a keyboard shortcut.
4. The user can highlight all icons on the page.

Furthermore, the software should satisfy the following non-functional requirements:

1. Labeling all icons on an average website should take no longer than 10 seconds.

The extension is developed to be used with the latest version (103) of the Google Chrome browser as of July 2022. Google Chrome is chosen as the target browsers as it features the highest worldwide market share of any web browser, with over 56.04% as of June 2022 according to SimilarWeb. As extensions follow the same core architecture across all modern browsers, the extension can be modified to be compatible with other browsers with minor changes.

## 6.2 Icon Detection

Before the semantic class of icons can be predicted, they first have to be detected inside the HTML document. In HTML, the key ways that icons can be implemented with is using either an SVG or image element or a text element in combination with an icon font [FMY+21]. The extension should be capable of detecting icons implemented with either of these methods.

While manually observing icons on popular webpages and their markup, the following heuristics are proposed to detect if an element is an icon:

1. The element is of type `img`, `svg` or `i`.
2. The element is smaller than $75 \times 75$ pixels.
3. The aspect ratio ($\frac{\text{width}}{\text{height}}$) is between 0.5 and 2.0.

**Listing 6.1** Sample implementation of the icon detection heuristics in the TypeScript programming language.

```
/**
 * findAllIcons returns all icons within the document.
 */
function findAllIcons() {
  const selector = "svg, img, i";
  const icons = [...document.querySelectorAll(selector)].filter((el) => {
    const aspectRatio = el.clientWidth / el.clientHeight;
    return (
      el.clientWidth <= 75 &&
      el.clientHeight <= 75 &&
      (aspectRatio >= 0.5 || aspectRatio <= 2)
    );
  });

  return icons
}
```

We define an element as an icon if it satisfies all the conditions stated above. Listing 6.1 shows a sample implementation of the heuristics in the TypeScript programming language.

### 6.2.1 Evaluation of the Heuristics

To evaluate the feasibility of the heuristics, they are evaluated on 10 popular real-world websites. The websites are identified by using their ranking in Germany on SimilarWeb. A full list of examined websites is shown in Appendix E.

First, the document of all 10 websites is exported as a static snapshot to avoid changes in layout or data between the different steps of the evaluation process. Next, all icons are manually identified on the page. A custom data attribute is added to the icon element inside the markup, indicating that the element is an icon. This custom data attribute serves the purpose of a marker to later reidentify the element. To avoid problems with pages that feature infinite scrolling lists, only icons in the initial viewport, i. e., icons that are visible without scrolling on a 1920 by 1080 pixel section of the page, are marked.

Now, an algorithm that applies the heuristics to every element of the site is executed. After the algorithm identifies possible icons, it automatically detects the presence of the custom data attribute and returns the results. The number of true positive, false positive and false negative detections are calculated. We define a true positive (TP) as an element that is an icon and is detected as an icon. A false positive (FP) is an element that is not an icon, but is detected as an icon. Finally, a false negative (FN) is an element that is an icon but is not detected as an icon. True negatives are omitted because they are not needed in any further calculations or evaluations. The results for TP, FP and FN are shown in Table 6.1. The precision, recall and F1-score are calculated:

| | | Predicted | |
|---|---|---|---|
| | | True | False |
| Actual | True | 119 | 8 |
| | False | 99 | - |

**Table 6.1:** Icon detection evaluation results.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{119}{119 + 99} \approx 0.5459 \tag{6.1}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{119}{119 + 8} \approx 0.9370 \tag{6.2}$$

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \approx 0.6899 \tag{6.3}$$

The results show that the system has a high recall. Recall describes the relative amount of icons that are correctly detected as icons, i. e., how many of the icons on the page are detected. A recall of 1 indicates, that all icons are detected by the algorithm, while a recall of 0 means that none of the icon are detected. Recall is particularly important for accessibility because it penalizes false negatives. False negatives, that is icons that are not detected by the heuristics, are to be avoided because they mean that automatically generated semantic labels are not available for some icons.

The precision score of the system is low. Precision describes how many of the as icons detected elements are actual icons. A precision of 1 means that all elements identified as icons are icons, while 0 equates to zero of the detected elements being icons. The score is low because a high number of false positives occur. We can trace back this issue to detected icons that are hidden from the user but are still present in the document and therefore visible to the algorithm. These icons may only show when the user hovers or scrolls over a specific element. This observation is particularly true on the examined page "Netflix.com"[1]. Here, the algorithm returned a total of 60 false positives and only 12 true positives. While false positives are not beneficial for the overall system and should be reduced, their presence does not impact the usability of the overall system from an accessibility standpoint. Even if additional elements are labeled that are not icons, all true positives, i. e., actual icons, will still be labeled. However, more time will have to be spent on the labeling of these non-icons, which slows down the overall system.

The $F_1$-score is a measure of the balance between precision and recall. Given the low precision and high recall, only a low F1-score is achieved.

We can observe the impact of recall and precision in Figure 6.1 which shows an example of the icon detection heuristics on YouTube. Detected icons are highlighted in red. We observe that most icons are correctly detected as icons. The recall of the system is high. The example, however, also

[1] https://www.netflix.com/

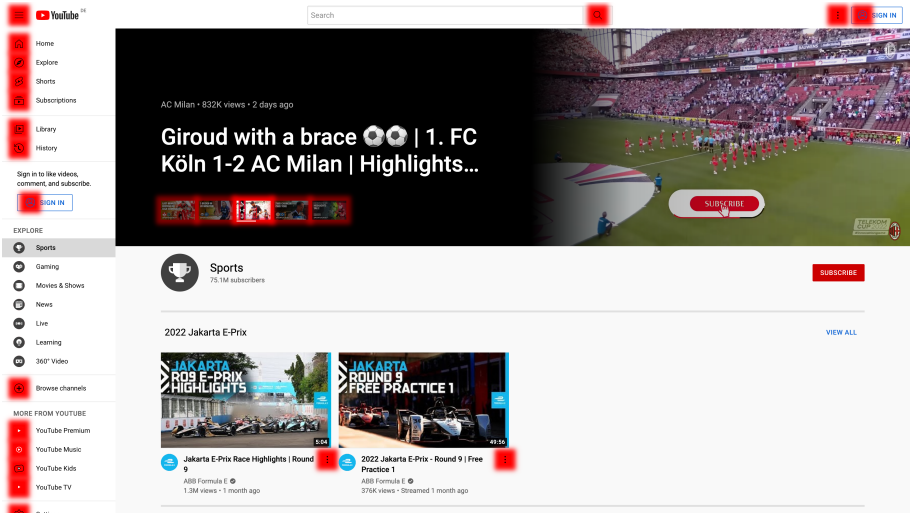


**Figure 6.1:** Example of detected icons on YouTube.

includes a number of small image thumbnails seen in the center of the page under the primary headline that are detected as icons. These thumbnails are classified as false positives and reduce the precision of the system.

We conclude that while the heuristics are not perfect, they are sufficient to reliably detect icons on webpages for the given use case. A lower precision is accepted in favor of a higher recall.

## 6.3 Implementation

The extension is implemented using the TypeScript programming language. TypeScript is a strict superset of JavaScript that provides a strong and statically checked type system. TypeScript compiles to JavaScript, which runs natively in any modern browser or server-side runtimes like Node.JS[2].

We utilize Tensorflow.js[3] to run the CNN developed in Chapter 5 directly inside the browser. Compared to a client-server model for classification, on-device inference in the browser has the advantage of preserving user privacy, since all data remains on the device. Tensorflow.js is a JavaScript implementation of TensorFlow. Tensorflow.js is capable of running a converted model of TensorFlow for Python and is therefore ideally suited for this use case. It features a similar API to the Keras API inside TensorFlow for Python. Using WebGL, Tensorflow.js is able to utilize hardware acceleration for fast inference speeds. Tensorflow.js has been used successfully in a variety of different tasks including image classification, object detection, speech command recognition and real-time pose detection[4].

[2] https://nodejs.org/
[3] https://www.tensorflow.org/js
[4] https://www.tensorflow.org/js/models

Browser extensions have the capabilities to hook into different events and to interact with the user and browser. Browser extensions consist of multiple components that interwork together. For this thesis, particularly important are the action and content script components. The action script is responsible for the interaction with the user by displaying a graphical user interface inside a popup in the browser's toolbar. It allows the user to trigger icon labeling on a page or highlight all detected icons. The content script is a JavaScript file that is dynamically injected into the webpage and runs within the context of that page. Therefore, content scripts have the capability to access to the document of the current page and carry out detections or modifications of elements. It is responsible for detecting and labeling the icons.

Icons are detected using the heuristics described and evaluated in Section 6.2. After icons have been identified, they have to be converted to image tensors, so they can be used as input to the neural network. As direct access to a screenshot API is not possible within an extensions content script, another approach is required to obtain the pixel values of an icon. We utilize the html2canvas[5] library to handle the conversion of an element inside the document to an image. Html2canvas allows rendering a wide variety of HTML element to a canvas element. Canvas elements are HTML elements that can be used to programmatically draw pixels and shapes on. They can be interpreted by Tensorflow.js as images.

After icons have been detected and converted to images, the neural network developed in Chapter 5 is used to predict the semantic classes of the icons. We convert the TensorFlow for Python model in Keras format to a Tensorflow.js model using tensorflow-converter[6]. Tensorflow-converter is capable of sharding the network, i. e., splitting it up into smaller chunks, that can then be loaded into the browser. Data augmentation layers detailed are disabled. To take advantage or our approach to labeling compound icons, we have to run inference twice for every icon. Once with the CNN that has been trained to detect the primary label, and once with the CNN that has been trained to detect the secondary label. After the primary and secondary semantic class of the icon are predicted, the two semantic classes are concatenated to form the final label. Finally, we programmatically attach the final label as an WAI-ARIA attribute to the icon. Figure 6.3 shows a visualization of this data flow throughout the extension.

The TypeScript source code is compiled to JavaScript and bundled with all its dependencies using the ESBuild[7] build tool. The output is a single JavaScript file for each the popup and content script components.

The implementation satisfies all functional requirements stated at the beginning of this chapter. Based on the fast inference time of the model, as evaluated in Section 5.6.3, we consider the non-functional regarding execution time to be satisfied.

An example screenshot of the extension running on a complex interactive website with many icons is shown in Figure 6.2. Figure 6.4 shows the accessibility tree, i. e., the descriptive name passed to a screen reader by the browser, for an icon after it has been labeled by the extension.

We acknowledge the following limitations and shortcoming of the current implementation:

1. Highly dynamic websites are not automatically relabeled when content changes.

[5] https://html2canvas.hertzen.com/

[6] https://github.com/tensorflow/tfjs/tree/master/tfjs-converter

[7] https://esbuild.github.io/

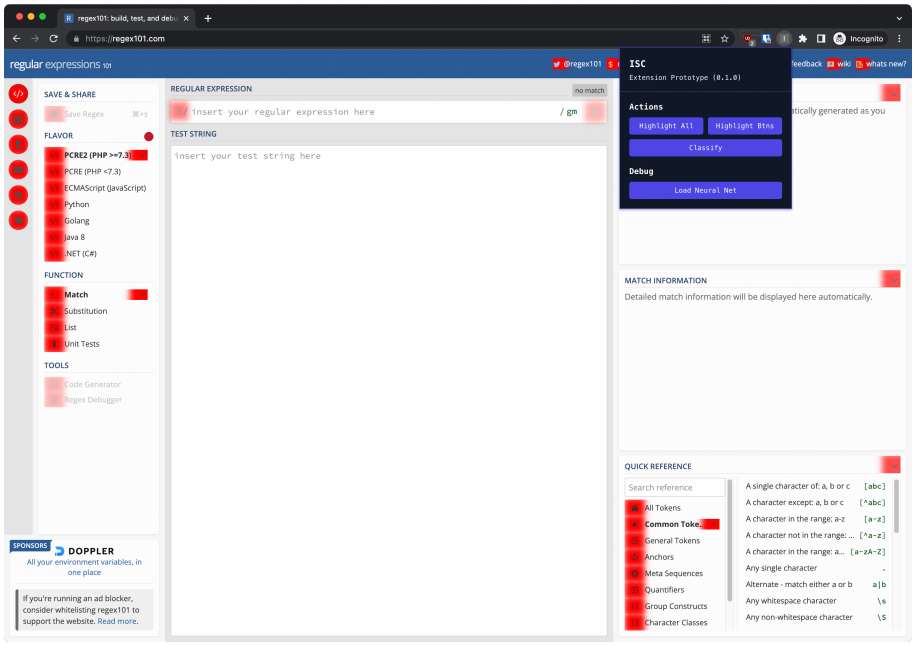


**Figure 6.2:** Screenshot of the browser extension running on a website with a large number of icons (`https://regex101.com/`).

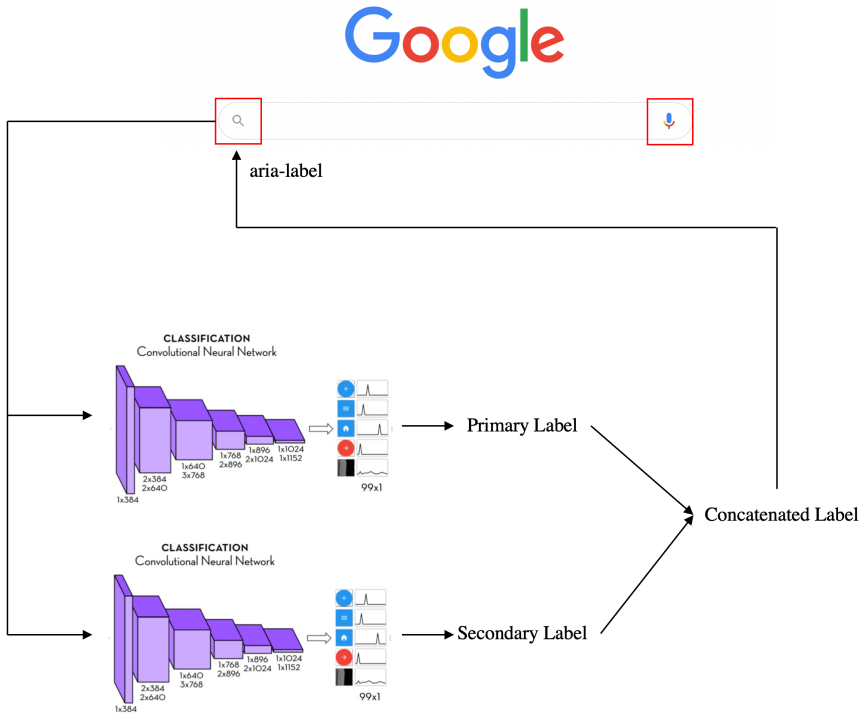


**Figure 6.3:** Dataflow within the browser extension. Images of CNN from Liu et al.[LCS+18].


**Figure 6.4:** Left: Screenshot of an icon of the semantic class “send”; Center: Markup of the icon annotated with an WAI-ARIA attribute by the browser extension; Right: Generated accessibility tree of the icon by the browser

2. Websites that make use of icons that are implemented using web fonts are not supported if they use any other element type than `i` for the icon element.

# 7 Conclusion

In this thesis, we improve the semantic icon classification approach by Liu et al. with respect to icons on the web. This is achieved by extending an existing dataset containing icons from mobile applications with icons from multiple sources of web icons, including open-source icon sets, icon databases and scraping icons from popular websites. We demonstrate that a CNN trained on the extended dataset features a 6% higher accuracy on a real-world web icon classification task compared to the same CNN being trained on the original dataset. Moreover, we propose a, to the best of our knowledge, novel approach to classifying compound icons. We achieve this by adding a second label to each icon, indicating the presence of a secondary icon. Using this approach, we are able to predict over five times as many semantic classes as Liu et al. We demonstrate the real-world practicality of the model by implementing a browser extension that automatically attaches missing accessibility metadata to icons and therefore helps people which are reliant on screen readers to perceive websites that were previously inaccessible to them. We introduce a set of heuristics that can be used to detect icons on a webpage and demonstrate their feasibility by evaluating their performance on popular websites.

Given the breadth of the topic and the work documented in this thesis, we see numerous opportunities for future enhancement, extension and application.

**Improving Classification Performance** Future work could further improve the classifier's performance on web icons. This could be achieved by collecting a large set of icons from real-world websites and adding them to the dataset. This work resorts to utilizing icons from websites only for the purpose of validation. Furthermore, an optional secondary input could be added to supply the classifier with contextual data, e. g., nearby located text. As our evaluation showed a low precision score on the semantic class "other", future work could opt to using an anomaly detection approach akin to Liu et al.[LCS+18] to detect images that are not icons.

**Compound Icons** While the classification approach developed in this thesis improves over related work by being capable of assigning semantic classes to a subset of compound icons, future work may extend on this topic by developing a classifier that is capable of assigning semantic classes to arbitrarily complex compound icons. One possible approach is to use object detection and image segmentation techniques to decompose the icon into its constituent parts. The classifier proposed in this work could be used to assign semantic classes to the individual parts of the composite icon. The entire semantic class can then be constructed from the individual parts.

**Large-scale study of icons on the web** Another topic of future research is further analysis of the different types and variety of icons on the Web. This could include further investigation of the number of composite icons on websites and how they are constructed. Also, the number

of websites that use the same icon sets could be studied. This might provide insight into how prevalent popular icon sets such as Material Design icons are on the Web. A large-scale study can be conducted using the icon extractor and the heuristics developed in Section 4.3.2.

Based on our findings during data analysis, we propose a further study of icon similarity within semantic classes using the deep learning icon similarity metric introduced by Lagunas et al.[LGG19].

**Applications** While the focus of this work lies on the topic of accessibility, there are other use cases that could benefit from the results of this thesis. Most prominently, a voice control system could use the ability to label icons on the web to provide voice access to previously inaccessible content by voice commands only. This is similar to Google IconNet[BS], which is however only available on Android devices.

Another use-case may include indexing of icon heavy websites for search engines. If a site features little or no machine-readable text, automatically generated icon descriptions could be used to gain further information about the websites content.

**Browser Extension** We are currently working on improving the usability and performance of the browser extension developed in Chapter 6. This includes more precise icon detection heuristics for icon elements implemented using fonts, automatic relabeling of icons on highly interactive websites and faster overall performance and memory efficiency.

# 8 Acknowledgement

I thank Prof. Dr. Steffen Staab for his early feedback during the intermediate presentation of this thesis and for his valuable suggestions. I thank my two advisors Ramin Hedeshy and Raphael Menges for their time, support and helpful advice and for providing me access to the necessary resources to complete this thesis.

# Bibliography


[AMA17] S. Albawi, T. A. Mohammed, S. Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186 (cit. on p. 14).

[AMJ+14] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, D. Yu. "Convolutional neural networks for speech recognition". In: *IEEE/ACM Transactions on audio, speech, and language processing* 22.10 (2014), pp. 1533–1545 (cit. on p. 16).

[Aph15] Aphex34. *Typical CNN architecture*. 2015. URL: https://commons.wikimedia.org/wiki/File:Typical_cnn.png (visited on 07/30/2022) (cit. on p. 16).

[BKL09] S. Bird, E. Klein, E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc, 2009 (cit. on p. 26).

[BS] G. Baechler, G. R. Srinivas Sunkara Software Engineers. *Improving Mobile App Accessibility with Icon Detection*. https://ai.googleblog.com/2021/01/improving-mobile-app-accessibility-with.html. Accessed: 2021-11-04 (cit. on pp. 9, 33, 52).

[BSG89] M. M. Blattner, D. A. Sumikawa, R. M. Greenberg. "Earcons and icons: Their structure and common design principles". In: *Human–Computer Interaction* 4.1 (1989), pp. 11–44 (cit. on pp. 5, 19, 22).

[Cam22] Cambridge-Dictionary. *Accessibility*. 2022. URL: https://dictionary.cambridge.org/dictionary/english/accessibility (visited on 07/02/2022) (cit. on p. 16).

[CCX+20] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, J. Wang. "Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. Seoul, South Korea: Association for Computing Machinery, 2020, pp. 322–334. ISBN: 9781450371216. DOI: 10.1145/3377811.3380327. URL: https://doi.org/10.1145/3377811.3380327 (cit. on pp. 5, 9, 11, 14, 16).

[DDS+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 38).

[DHF+17] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, R. Kumar. "Rico: A Mobile App Dataset for Building Data-Driven Design Applications". In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. UIST '17. Québec City, QC, Canada: Association for Computing Machinery, 2017, pp. 845–854. ISBN: 9781450349819. DOI: 10.1145/3126594.3126651. URL: https://doi.org/10.1145/3126594.3126651 (cit. on pp. 9, 25).

[FMY+21] S. Feng, S. Ma, J. Yu, C. Chen, T. Zhou, Y. Zhen. "Auto-Icon: An Automated Code Generation Tool for Icon Designs Assisting in UI Development". In: *26th International Conference on Intelligent User Interfaces*. IUI '21. College Station, TX, USA: Association for Computing Machinery, 2021, pp. 59–69. ISBN: 9781450380171. DOI: 10.1145/3397481.3450671. URL: https://doi.org/10.1145/3397481.3450671 (cit. on pp. 10, 11, 33, 43).

[Git86] D. Gittins. "Icon-based human-computer interaction". In: *International Journal of Man-Machine Studies* 24.6 (1986), pp. 519–543 (cit. on pp. 5, 19).

[GSF01] R. S. Goonetilleke, H. M. Shih, J. Fritsch. "Effects of training and representational characteristics in icon design". In: *International Journal of Human-Computer Studies* 55.5 (2001), pp. 741–760 (cit. on pp. 19, 20).

[GWK+18] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, T. Chen. "Recent advances in convolutional neural networks". In: *Pattern Recognition* 77 (2018), pp. 354–377. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.10.013. URL: https://www.sciencedirect.com/science/article/pii/S0031320317304120 (cit. on pp. 11, 16, 34).

[HAE16] M. Huh, P. Agrawal, A. A. Efros. "What makes ImageNet good for transfer learning?" In: *CoRR* abs/1608.08614 (2016). arXiv: 1608.08614. URL: http://arxiv.org/abs/1608.08614 (cit. on p. 38).

[HZC+17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017) (cit. on p. 33).

[KB15] D. P. Kingma, J. Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio, Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 38).

[LCS+18] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, R. Kumar. "Learning Design Semantics for Mobile Apps". In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. UIST '18. Berlin, Germany: Association for Computing Machinery, 2018, pp. 569–579. ISBN: 9781450359481. DOI: 10.1145/3242587.3242650. URL: https://doi.org/10.1145/3242587.3242650 (cit. on pp. 5, 6, 9, 11, 16, 22–25, 28, 33, 36, 40, 41, 48, 51).

[LGG19] M. Lagunas, E. Garces, D. Gutierrez. "Learning icons appearance similarity". In: *Multimedia tools and applications* 78.8 (2019), pp. 10733–10751 (cit. on pp. 6, 27, 29, 30, 32, 52).

[MAP+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin

Wicke, Yuan Yu, Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (cit. on p. 36).

[MC04] S. McDougall, M. Curry. "More than just a picture: Icon interpretation in context". In: *Proceedings of the First International Workshop on Coping with Complexity*. Citeseer. 2004, pp. 73–81 (cit. on p. 19).

[MCD99] S. J. Mcdougall, M. B. Curry, O. De Bruijn. "Measuring symbol and icon characteristics: Norms for concreteness, complexity, meaningfulness, familiarity, and semantic distance for 239 symbols". In: *Behavior Research Methods, Instruments, & Computers* 31.3 (1999), pp. 487–519 (cit. on pp. 5, 19).

[Mil95] G. A. Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41 (cit. on p. 26).

[ON15] K. O'Shea, R. Nash. "An Introduction to Convolutional Neural Networks". In: *CoRR* abs/1511.08458 (2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458 (cit. on pp. 11, 14–16).

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 36).

[RN09] S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597 (cit. on p. 11).

[Rud16] S. Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (cit. on p. 14).

[RZFW18] A. S. Ross, X. Zhang, J. Fogarty, J. O. Wobbrock. "Examining Image-Based Button Labeling for Accessibility in Android Apps through Large-Scale Analysis". In: *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. ASSETS '18. Galway, Ireland: Association for Computing Machinery, 2018, pp. 119–130. ISBN: 9781450356503. DOI: 10.1145/3234695.3236364. URL: https://doi.org/10.1145/3234695.3236364 (cit. on p. 5).

[SHZ+18] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, L. Chen. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381 (cit. on pp. 14, 33–35).

[Ten22] TensorFlow. *Transfer learning and fine-tuning*. 2022. URL: https://www.tensorflow.org/tutorials/images/transfer_learning (visited on 05/03/2022) (cit. on pp. 36, 38).

[VH08] L. Van der Maaten, G. Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008) (cit. on p. 30).

[WSD+17] C. Wigington, S. Stewart, B. Davis, B. Barrett, B. Price, S. Cohen. "Data Augmentation for Recognition of Handwritten Words and Lines Using a CNN-LSTM Network". In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 01. 2017, pp. 639–645. DOI: 10.1109/ICDAR.2017.110 (cit. on p. 36).

[XWC+19] X. Xiao, X. Wang, Z. Cao, H. Wang, P. Gao. "IconIntent: Automatic Identification of Sensitive UI Widgets Based on Icon Classification for Android Apps". In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 2019, pp. 257–268. DOI: 10.1109/ICSE.2019.00041 (cit. on p. 9).

[XYX+19] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, J. Lu. "DeepIntent: Deep Icon-Behavior Learning for Detecting Intention-Behavior Discrepancy in Mobile Apps". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 2421–2436. ISBN: 9781450367479. DOI: 10.1145/3319535.3363193. URL: https://doi.org/10.1145/3319535.3363193 (cit. on pp. 10, 11, 20, 21).

[YNDT18] R. Yamashita, M. Nishio, R. K. G. Do, K. Togashi. "Convolutional neural networks: an overview and application in radiology". In: *Insights into imaging* 9.4 (2018), pp. 611–629 (cit. on p. 16).

[ZKH+20] N. Zhao, N. W. Kim, L. M. Herman, H. Pfister, R. W. Lau, J. Echevarria, Z. Bylinskii. "ICONATE: Automatic Compound Icon Generation and Ideation". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1–13. ISBN: 9781450367080. URL: https://doi.org/10.1145/3313831.3376618 (cit. on p. 22).

[ZXC21] X. Zang, Y. Xu, J. Chen. "Multimodal Icon Annotation For Mobile Applications". In: *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction*. MobileHCI 21. Toulouse & Virtual, France: Association for Computing Machinery, 2021. ISBN: 9781450383288. DOI: 10.1145/3447526.3472064. URL: https://doi.org/10.1145/3447526.3472064 (cit. on pp. 5, 9, 20, 21).

# A List of Semantic Classes

List of all primary 113 semantic classes sorted alphabetically in ascending order:

- add
- archive
- arrow_b
- arrow_bl
- arrow_br
- arrow_l
- arrow_r
- arrow_t
- arrow_tl
- arrow_tr
- attach
- av_forward
- av_rewind
- avatar
- bluetooth
- book
- bookmark
- build
- call
- camera
- cart
- chart
- chat
- check
- close

- compare
- copy
- currency
- dashboard
- date
- delete
- description
- dialpad
- download
- edit
- email
- emoji
- expand_less
- expand_more
- explore
- facebook
- file
- filter
- flash
- folder
- font
- fullscreen
- gift
- globe
- group
- heart
- help
- history
- home
- image
- info

- key
- label
- language
- launch
- layers
- link
- list
- location
- location_crosshair
- lock
- login
- logout
- map
- menu
- microphone
- minus
- more
- move
- music
- national_flag
- navigation
- network_wireless
- notifications
- other
- pause
- play
- playlist
- power
- redo
- refresh
- repeat

- reply
- save
- search
- send
- settings
- share
- shop
- skip_next
- skip_previous
- star
- stop
- swap
- switcher
- thumbs_down
- thumbs_up
- time
- twitter
- undo
- upload
- videocam
- visibility
- volume
- warning
- weather
- zoom_in
- zoom_out

List of all 6 secondary semantic classes sorted alphabetically in ascending order:

- 2add
- check
- delete
- edit

- other
- search

# B Observed Websites for Assumptions on Compound Icons

- Google.com (https://www.google.com/)
- YouTube.com (https://www.youtube.com/)
- Facebook.com (https://www.facebook.com/)
- Amazon.de (https://www.amazon.de/)
- Wikipedia.org (https://www.wikipedia.org/)
- Bild.de (https://www.bild.de/)
- Ebay.de (https://www.ebay.de/)
- Instagram.com (https://www.instagram.com/)
- Ebay Kleinanzeigen (https://www.ebay-kleinanzeigen.de/)
- T-Online.de (https://www.t-online.de/)
- Twitter.com (https://www.twitter.com/)
- Stackoverflow.com (https://www.stackoverflow.com/)
- Twitch.tv (https://www.twitch.tv/)
- npmjs.com (https://www.npmjs.com/)
- Github.com (https://www.github.com/)

# C Websites Used for Icon Extraction

- Google.com (https://www.google.com/)
- YouTube.com (https://www.youtube.com/)
- Facebook.com (https://www.facebook.com/)
- Amazon.de (https://www.amazon.de/)
- Wikipedia.org (https://www.wikipedia.org/)
- Ebay.de (https://www.ebay.de/)
- Instagram.com (https://www.instagram.com/)
- Ebay Kleinanzeigen (https://www.ebay-kleinanzeigen.de/)
- Twitter.com (https://www.twitter.com/)
- Stackoverflow.com (https://www.stackoverflow.com/)
- Regex101.com (https://regex101.com/)
- LinkedIn.com (https://www.linkedin.com/)
- Pinterest.com (https://www.pinterest.com/)
- Reddit.com (https://www.reddit.com/)
- Tumblr.com (https://www.tumblr.com/)
- Tiktok.com (https://www.tiktok.com/)
- Twitch.tv (https://www.twitch.tv/)
- npmjs.com (https://www.npmjs.com/)
- Github.com (https://www.github.com/)

# D Precision, Recall and F1 Score for Each Semantic Class and Confusion Matrix

Table D.1 shows the precision, recall and F1 score for every primary semantic class.

Table D.1 shows the precision, recall and F1 score for every secondary semantic class.

The confusion matrix for the primary semantic classes is shown in Figure D.1 and the confusion matrix for the secondary semantic classes is shown in Figure D.2.

| Semantic Class | Precision | Recall | F1 |
|---|---|---|---|
| bluetooth | 0.996 | 0.987 | 0.991 |
| volume | 0.979 | 0.983 | 0.981 |
| lock | 0.967 | 0.992 | 0.979 |
| more | 0.982 | 0.977 | 0.979 |
| bookmark | 0.983 | 0.975 | 0.979 |
| gift | 0.996 | 0.961 | 0.978 |
| arrow_l | 0.986 | 0.969 | 0.978 |
| filter | 0.992 | 0.961 | 0.976 |
| home | 0.978 | 0.970 | 0.974 |
| pause | 0.957 | 0.992 | 0.974 |
| network_wireless | 0.983 | 0.963 | 0.973 |
| dashboard | 0.983 | 0.962 | 0.972 |
| switcher | 0.995 | 0.949 | 0.972 |
| power | 0.963 | 0.979 | 0.971 |
| arrow_tl | 0.943 | 1.000 | 0.971 |
| shop | 0.973 | 0.962 | 0.967 |
| menu | 0.969 | 0.964 | 0.966 |
| flash | 0.964 | 0.968 | 0.966 |
| group | 0.959 | 0.969 | 0.964 |
| dialpad | 0.935 | 0.995 | 0.964 |
| thumbs_down | 0.965 | 0.962 | 0.964 |
| layers | 0.978 | 0.949 | 0.963 |
| emoji | 0.950 | 0.972 | 0.961 |
| font | 0.968 | 0.953 | 0.961 |
| microphone | 0.981 | 0.941 | 0.960 |
| close | 0.970 | 0.950 | 0.960 |
| launch | 0.937 | 0.982 | 0.959 |

| | | | |
|---|---|---|---|
| visibility | 0.975 | 0.943 | 0.959 |
| globe | 0.960 | 0.955 | 0.957 |
| build | 0.947 | 0.967 | 0.957 |
| star | 0.974 | 0.939 | 0.956 |
| notifications | 0.970 | 0.942 | 0.956 |
| image | 0.937 | 0.975 | 0.955 |
| search | 0.956 | 0.952 | 0.954 |
| music | 0.979 | 0.931 | 0.954 |
| save | 0.981 | 0.929 | 0.954 |
| stop | 0.935 | 0.972 | 0.953 |
| videocam | 0.931 | 0.975 | 0.953 |
| av_rewind | 0.954 | 0.950 | 0.952 |
| explore | 0.948 | 0.951 | 0.950 |
| national_flag | 0.949 | 0.949 | 0.949 |
| attach | 0.942 | 0.953 | 0.948 |
| fullscreen | 0.978 | 0.918 | 0.947 |
| warning | 0.951 | 0.942 | 0.947 |
| expand_more | 0.938 | 0.954 | 0.946 |
| add | 0.974 | 0.919 | 0.946 |
| book | 0.956 | 0.935 | 0.946 |
| edit | 0.958 | 0.932 | 0.944 |
| weather | 0.954 | 0.934 | 0.944 |
| call | 0.971 | 0.915 | 0.942 |
| thumbs_up | 0.912 | 0.972 | 0.941 |
| date | 0.912 | 0.970 | 0.940 |
| check | 0.959 | 0.921 | 0.939 |
| undo | 0.948 | 0.928 | 0.938 |
| location_crosshair | 0.919 | 0.956 | 0.937 |
| heart | 0.948 | 0.926 | 0.937 |
| folder | 0.930 | 0.941 | 0.935 |
| email | 0.914 | 0.957 | 0.935 |
| info | 0.946 | 0.921 | 0.934 |
| settings | 0.903 | 0.958 | 0.930 |
| skip_previous | 0.955 | 0.906 | 0.930 |
| playlist | 0.968 | 0.891 | 0.928 |
| help | 0.962 | 0.894 | 0.927 |
| camera | 0.898 | 0.957 | 0.927 |
| history | 0.949 | 0.904 | 0.926 |
| send | 0.966 | 0.886 | 0.924 |
| skip_next | 0.888 | 0.961 | 0.923 |
| share | 0.948 | 0.893 | 0.920 |

| | | | |
|---|---|---|---|
| download | 0.928 | 0.906 | 0.917 |
| cart | 0.906 | 0.928 | 0.917 |
| location | 0.939 | 0.895 | 0.916 |
| reply | 0.922 | 0.905 | 0.914 |
| navigation | 0.905 | 0.922 | 0.913 |
| repeat | 0.915 | 0.911 | 0.913 |
| arrow_b | 0.913 | 0.913 | 0.913 |
| redo | 0.915 | 0.902 | 0.908 |
| chat | 0.924 | 0.891 | 0.907 |
| list | 0.869 | 0.942 | 0.904 |
| twitter | 0.928 | 0.876 | 0.902 |
| expand_less | 0.895 | 0.903 | 0.899 |
| arrow_t | 0.907 | 0.890 | 0.898 |
| copy | 0.862 | 0.936 | 0.898 |
| compare | 0.891 | 0.898 | 0.895 |
| play | 0.892 | 0.896 | 0.894 |
| swap | 0.870 | 0.918 | 0.894 |
| time | 0.892 | 0.892 | 0.892 |
| delete | 0.879 | 0.899 | 0.889 |
| label | 0.839 | 0.941 | 0.887 |
| arrow_r | 0.904 | 0.870 | 0.887 |
| description | 0.850 | 0.895 | 0.872 |
| refresh | 0.824 | 0.923 | 0.871 |
| avatar | 0.884 | 0.856 | 0.870 |
| minus | 0.794 | 0.939 | 0.860 |
| facebook | 0.917 | 0.806 | 0.858 |
| av_forward | 0.855 | 0.851 | 0.853 |
| zoom_out | 0.899 | 0.798 | 0.845 |
| link | 0.761 | 0.833 | 0.795 |
| logout | 0.800 | 0.727 | 0.762 |
| key | 0.679 | 0.844 | 0.752 |
| move | 0.833 | 0.638 | 0.723 |
| currency | 0.733 | 0.673 | 0.702 |
| map | 0.610 | 0.806 | 0.694 |
| upload | 0.659 | 0.674 | 0.667 |
| file | 0.516 | 0.800 | 0.627 |
| zoom_in | 0.800 | 0.500 | 0.615 |
| archive | 0.567 | 0.586 | 0.576 |
| login | 0.613 | 0.543 | 0.576 |
| arrow_tr | 0.500 | 0.600 | 0.545 |
| language | 0.571 | 0.513 | 0.541 |

| | | | |
|---|---|---|---|
| arrow_bl | 0.450 | 0.600 | 0.514 |
| chart | 0.731 | 0.365 | 0.487 |
| other | 0.316 | 0.820 | 0.457 |
| arrow_br | 0.375 | 0.400 | 0.387 |

**Table D.1:** Precision, recall and F1 Score for each primary semantic class sorted by F1 score in descending order.

| Semantic Class | Precision | Recall | F1 |
|---|---|---|---|
| other | 0.962 | 0.997 | 0.979 |
| search | 0.996 | 0.959 | 0.977 |
| delete | 0.987 | 0.963 | 0.975 |
| add | 0.990 | 0.930 | 0.959 |
| edit | 0.974 | 0.917 | 0.945 |
| check | 0.980 | 0.879 | 0.927 |

**Table D.2:** Precision, recall and F1 Score for each secondary semantic class sorted by F1 score in descending order.

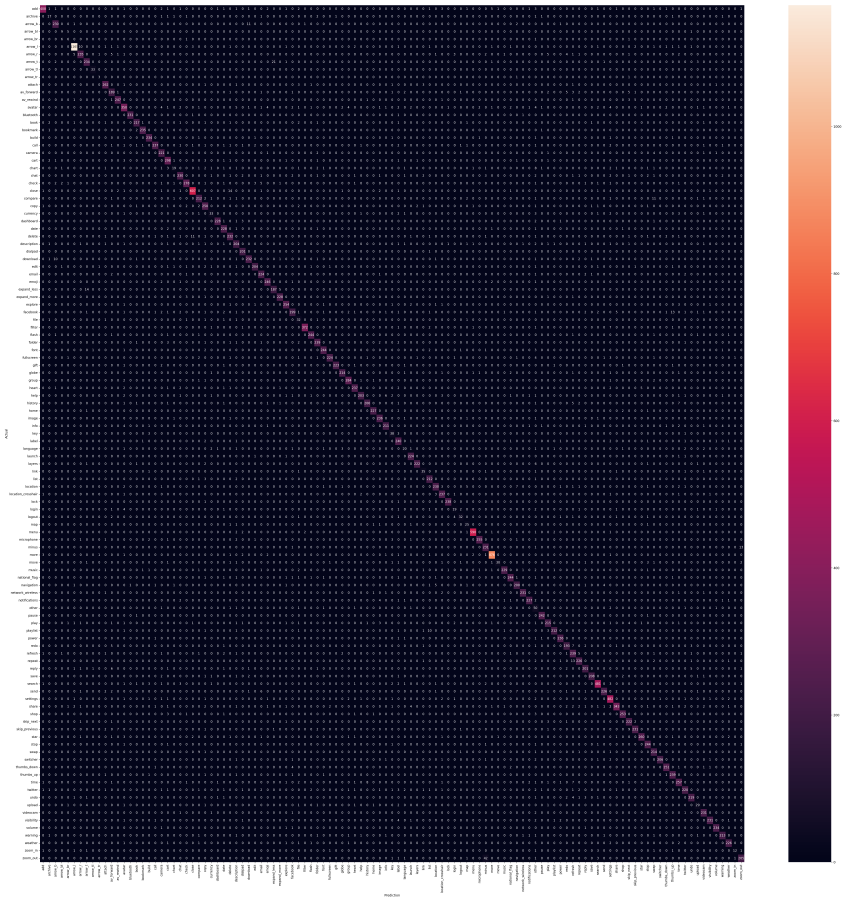

**Figure D.1:** Confusion matrix of all primary semantic classes.

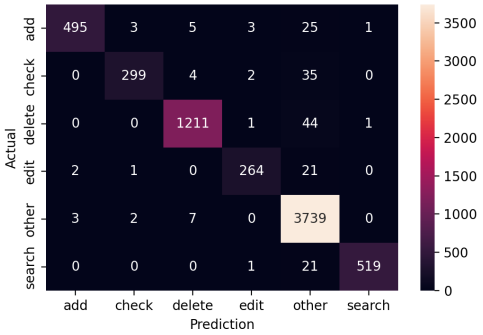


**Figure D.2:** Confusion matrix of all secondary semantic classes.

# E Examined Websites During Icon Detection Heuristics Evaluation

- Google.com (https://www.google.com/)
- YouTube.com (https://www.youtube.com/)
- Facebook.com (https://www.facebook.com/)
- Amazon.de (https://www.amazon.de/)
- Wikipedia.org (https://www.wikipedia.org/)
- Ebay.de (https://www.ebay.de/)
- Instagram.com (https://www.instagram.com/)
- Ebay Kleinanzeigen (https://www.ebay-kleinanzeigen.de/)
- Twitter.com (https://www.twitter.com/)
- Netflix.com (https://www.netflix.com/)

## Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift:

## Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: