Uma Extensão do Chrome para Filtragem de Imagens Digitais Usando Redes Neurais Convolucionais

1st Samuel Cavalcanti *Escola de Ciências e Tecnologia*
Universidade Federal do Rio Grande do Norte (UFRN)
Natal, Brasil
scavalcanti111@gmail.com

2nd Agostinho De Medeiros Brito Junior
Departamento de Engenharia de Computação e Automação (DCA)
Universidade Federal do Rio Grande do Norte (UFRN)
Natal, Brasil
agostinhobritojr@gmail.com

21 de Outubro de 2020

Resumo

Trata de uma extensão de código aberto para navegadores baseados no motor do Google Chrome destinada à filtragem de imagens de cunho indesejado. A extensão pode ser configurada de modo que, caso uma determinada imagem de uma página web pertença a uma categoria que o usuário não queira observar (ex: imagens de conteúdo adulto), o conteúdo indesejado pode ser bloqueado. O processo de filtragem se dá pela implementação direta de uma rede neural convolucional treinada para classificar que combinem com um padrão configurado. Para garantir eficiência em tempo de resposta foi utilizado o framework em linguagem JavaScript de aprendizado de máquina tensorflow.js. Testes de eficiências foram realizados com redes pré-treinadas da MobileNet V2 e Inception V1 no conjunto de dados da ImageNet (ILSVRC-2012-CLS). Os resultados mostraram que utilizando um processador AMD Ryzen 5 2500u, o tempo de resposta em média das redes neurais foram inferiores a 5 milissegundos, mas a estratégia usada para enviar as imagens para rede aumenta em média 294.15 milissegundos e pode variar dependendo da velocidade da internet. Foi observada também alguma latência oriunda do aumento do uso de CPU no navegador em momentos que número de imagens requisitadas era elevado. Ainda assim, a filtragem é bem sucedida e mostra o sucesso da ferramenta implementada, principalmente quando esse tipo de operação de filtragem é mandatória para o usuário.

1 Introdução

Trata de uma extensão de código aberto para navegadores baseados no motor do Google Chrome destinada à filtragem de imagens de cunho indesejado. A extensão pode ser configurada de modo que, caso uma determinada imagem de uma página web pertença a uma categoria que o usuário não queira observar (ex: imagens de conteúdo adulto), o conteúdo indesejado pode ser bloqueado.

O processo de filtragem se dá pela implementação direta de uma rede neural convolucional treinada para classificar que combinem com um padrão configurado. Para garantir eficiência em tempo de resposta foi utilizado o framework em linguagem JavaScript de aprendizado de máquina tensorflow.js. Testes de eficiências foram realizados com redes pré-treinadas da MobileNet V2 e Inception V1 no conjunto de dados da ImageNet (ILSVRC-2012-CLS). Os resultados mostraram que utilizando um processador AMD Ryzen 5 2500u, o tempo de resposta em média das redes neurais foram inferiores a 5 milissegundos, mas a estratégia usada para enviar as imagens para rede aumenta em média 294.15 milissegundos e pode variar dependendo da velocidade da internet. Foi observada também alguma latência oriunda do aumento do uso de CPU no navegador em momentos que número de imagens requisitadas era elevado. Ainda assim, a filtragem é bem sucedida e mostra o sucesso da ferramenta implementada, principalmente quando esse tipo de operação de filtragem é mandatória para o usuário.É importante lembrar que o foco desse trabalho reside na implementação na extensão chrome que irá usar os modelos treinado, a acurácia das redes convolucionais apesar de bastante impactante na experiência do usuário, não sera um fator a ser considerado uma vez que esse trabalho é relato sobre o desenvolvimento de uma extensão com redes neurais convolucionais e tem como objetivo verificar a viabilidade de utilizar redes neurais convolucionais para filtragem de imagens em relação ao tempo de resposta dessa técnica, a ultima sessão desse artigo visa concluir sobre o tempo de resposta dessa extensão para exibir ou não uma imagem.

2 O funcionamento de uma Extensão chrome

Extensões para navegadores baseados no projeto chromium são construídos usando os seguintes componentes: background page ou script, content scripts, options page, elementos de interface gráfica e um arquivo de manifesto - *manifest.json* - onde ficam armazenados metadados da extensão. A extensão desenvolvida foi criada com esses vários componentes, muito embora em alguns casos, a presença de todos eles não seja necessária.

Toda a extensão Chrome possui um arquivo de manifesto. E nele que é informado ao navegador todas as permissões que a extensão necessita, o nome da extensão, uma breve descrição do que ela é, quais componentes serão utilizados nessa extensão e seus respectivos arquivos, entre outros metadados.

Toda Extensão pode possuir uma interação com usuário, essa interação acontece ao usuário clicar no ícone da extensão, dependendo da implementação o ato de clicar é toda interação com o usuário que aplicação pode ter ou que é mais comum é abrir uma página HTML comumente chamada de popup com todas as informações sua aplicação. Dependendo da página atual que o usuário está acessando é possível personalizar qual página HTML será exibida para o usuário ou até mesmo impedir do usuário de acessar as funcionalidades da extensão. [1]

Caso necessite que o usuário modifique ou personalize o comportamento da extensão, o desenvolvedor pode implementar uma options page que é uma página HTML listando todas as funcionalidades. Para acessar a options page o usuário precisa ir em *chrome://extensions*, localizar a extensão desejada, clicar em *Details* e clicar em *Extension options*.

Content scripts são arquivos que são executados em páginas web durante a navegação, esse arquivos são capazes de ler, modificar e passar informações da página atual para extensão, esses arquivos são executados isoladamente não conflitando entre si e nem com outros *scripts* presentes na página web apesar deles terem acesso ao mesmo *Document Object Model* (DOM).

O Responsável por monitorar os eventos do navegador é o background script ou background page. Ele é inicializado sempre que a extensão for instalada, atualizada ou quando acontece um evento a qual ele esteja monitorando, por exemplo, ao receber uma mensagem de outro componente ou quando uma determinada página começará a ser carregada. O *background script* só fica ativo quando estiver executando alguma ação. Assim que todas as ações forem finalizadas o *script* também é finalizado.

3 Rede neural convolucional

Uma Rede Neural Convolucional (*Convolutional Neural Network - CNN*) é uma variação das redes de Perceptrons de Múltiplas Camadas, tendo sido inspirada no processo biológico de processamentos de dados visuais. De maneira semelhante aos processos tradicionais de visão computacional, uma CNN é capaz de aplicar filtros em dados visuais, mantendo a relação de vizinhança entre os pixels de uma imagem ao longo do processamento da rede [2].

Esse trabalho apresenta uma extensão chrome possui as seguintes funcionalidades:

- lista todos os modelos de CNNs do portal *https://tfhub.dev/*, dedicado a hospedar diversas estruturas pré-treinadas para possíveis utilizadores. Os modelos presentes nesse portal são adaptados para classificação de imagens e possuem total compatibilidade com a ferramenta de classificação tensorflow.js. Na configuração da extensão, o usuário pode selecionar e usar um dos modelos Íá presentes.
- Exibe as classes que o modelo é capaz de classificar e permite o usuário selecionar qual classes ele quer ver ou não.
- Filtra as imagens com base no modelo que o usuário selecionou.
- carregar um modelo personalizado localmente.

Apesar de a extensão ser capaz de tratar todas os modelos, ainda não é possível o armazenamento de modelos personalizados carregados localmente, uma vez que isto demanda o armazenamento local de um grande volume de dados e não existe suporte para esse fim no ferramental disponibilizado para programação de extensões.

4 Funcionamento da extensão proposta

O objetivo da aplicação é monitorar todas as imagens recebidas pelo navegador e não exibir as imagens que o usuário não desejaria visualizar. Para alcançar esse objetivo, o problema foi fragmentado em um conjunto de problemas menores, implementados em interfaces, onde cada interface é especializada em um desses problemas e tenta prover uma boa solução.

Foram proposta as seguintes interfaces: gerenciador de classificadores, classificador, interface gráfica, observador de página, carregador de imagens, navegador-comunicador e registrador (*logger*) de tempo. O diagrama de interfaces e comunicações entre elas é ilustrado na Figura 1.

Os frameworks de desenvolvimento tensorflow.js e Angular framework foram usados para prover recursos de alto nível e permitir o foco na implementação da interação entre o navegador e a própria rede neural convolucional. Angular é um framework JavaScript de código aberto com o objetivo de

criar modernas aplicações web, mobile e desktop [3]. Por outro lado, tensorflow.js é uma biblioteca para construção e execução de algoritmos de aprendizado de máquina em JavaScript [4].

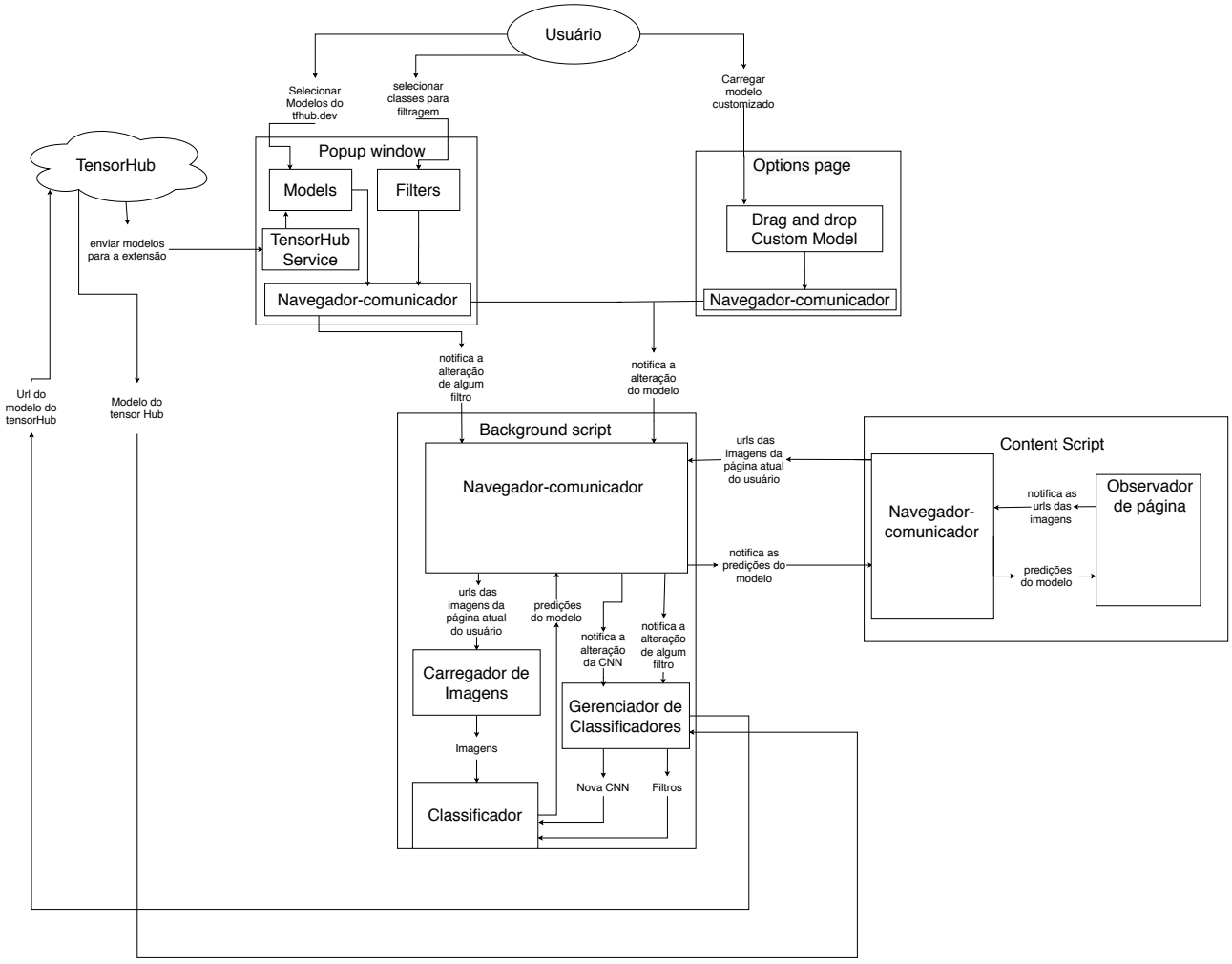


Figura 1: Diagrama da aplicação

O framework angular é essencial para aplicação, por a partir dele foram feitas modificações para, ao invés de compilar um front-end de uma aplicação web, o resultado da compilação ser uma extensão chrome. Para isso foram feitas as seguintes modificações:

- Foi adicionado uma configuração adicional no webpack que possibilita compilar os arquivos do content script.
- Foi habilitado o uso de hash no caminho nas rotas.
- Observando os arquivos resultantes da compilação e minificação do webpack, adicionou-se os seus nomes ao arquivo `manifest.json`.

Para adicionar a configuração adicional no webpack, foi editado o arquivo `angular.json` (trecho mostrado na Listagem 1). Nesse arquivo é onde ficam armazenados os metadados do projeto angular, configurações de build do projeto, inclusive um caminho para uma configuração personalizadada

do webpack ((trecho mostrado na Listagem 2). No arquivo de configuração do webpack foi adicionado o arquivo `content-script.ts`, que contém todas as rotinas de recuperação de URLs das imagens e envio para o background page. Este, por sua vez, retorná a ação de exibir a imagem ou deixá-la escondida.

Listagem 1 Trecho de configuração no Angular

```
"build": {
  "builder": "@angular-builders/custom-webpack:browser",
  "options": {
      "customWebpackConfig": {
      "path": "./custom-webpack.config.js"
   },
```

Listagem 2 Trecho de configuração no webpack

```
module.exports = {
   entry: {'content-script': 'src/content-script/content-script.ts'},
}
```

Outra alteração importante foi habilitar o uso de hash no caminho das rotas (ver Listagens 3 e 4, pois através dessa configuração é possível utilizar um único arquivo HTML tanto para o background script, a janela de popup e options page desde cada um dos componentes tenha a sua própria rota. Depois de transformar um projeto angular em um projeto capaz de produzir uma extensão, foi feita a modularização do projeto com o objetivo de reaproveitar ainda mais o código e ser capaz de configurar o lazy-loading nas rotas para otimizar o carregamento do código, dessa forma durante a execução do background script, os código das da interface gráfica não serão carregados, aumentando ainda mais a performance da extensão, pois além do código JavaScript gerado ser minificado e reduzido o lazy-loading possibilita que o código gerado por uma das rotas não seja carregado inicialmente.

4.1 Notificação

A notificação é a interface mais simples e mais utilizada em em toda a aplicação. Ela é uma interface que dita o formato da informação que ser trafegada entre as interfaces, quando, por exemplo, a interface gráfica precisa transmitir uma informação para o background script, se essa informação será passada no formato de notificação.

4.2 Navegador-Comunicador

Com o objetivo de abstrair o navegador surgiu essa interface. O navegador-comunicador é responsável para realizar qualquer chamada da API do navegador Chrome. É através dele que diferentes componentes da extensão se comunicam, como a interface gráfica com o background script, ou o *background script* com o *content script*, *options page* com background script, além também de ser responsável por armazenar informações no navegador e recuperar arquivos que foram instalados juntos com a extensão.

Listagem 3 Arquivo de rotas

```
import {NgModule} from "@angular/core"
import {RouterModule, Routes} from "@angular/router"
import {BackgroundComponent} from "./background-page/background.component"


const routes: Routes = [
    {path: "filters", loadChildren: () => import("./filters-page/filters.module")
    .then(m => m.FiltersModule)},
    {path: "models", loadChildren: () => import("./models-page/cnn-models.module")
    .then(m => m.CnnModelsModule)},
    {path: "options", loadChildren: () => import("./options-page/options-page.module
        ")
    .then(m => m.OptionsPageModule)},
    {path: "background", component: BackgroundComponent},
]

@NgModule({
    imports: [RouterModule.forRoot(routes, {useHash: true})], // <-- uso de hash
        habilitado
    exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Listagem 4 Efeitos da configuração de rotas Angular no Manifest.json

```
"background": {
   "page": "index.html#/background",
   "persistent": true
},
    "web_accessible_resources": [
    "assets/modelJS/*"
    ],
    "browser_action": {
    "default_popup": "index.html",
```

Uma das principais motivações da criação dessa interface é isolar as funcionalidades particulares do navegador Chrome, para permitir futura portabilidade para outro navegador. Assim, o código atual não precisaria sofrer grandes modificações.

4.3 Observador de página

Essa interface é responsável por monitorar as páginas que o usuário está acessando e notificar para outros componentes todas as URLs de imagens encontradas. Ele espera ser notificado se as imagens notificadas podem ou não ser visualizadas. Esse componente se localiza exclusivamente no content script da nossa aplicação, uma vez que é no content script que se tem acesso as páginas acessadas pelo usuário.

4.4 Interface gráfica

Durante o desenvolvimento dessa aplicação foi construído uma página HTML que é exibida assim que o usuário clica no ícone da extensão. Essa página se comporta como uma single page web application. Uma single page web application é exatamente o que o nome diz, uma aplicação web feita em JavaScript que necessita o carregamento de uma única página [5], no entanto essa página pode ter diferentes sub-páginas. Através dessa página HTML o usuário pode selecionar um modelo do *https://tfhub.dev/* e também marcar quais classes de imagens ele deseja ou não visualizar, por padrão ao escolher um modelo você pode visualizar todas as classes de imagens, como ilustram as Figuras 2 e 3. Também é nessa interface que o usuário pode inserir um modelo personalizado porém, como essa opção é para usuários avançados foi decidido colocá-lo no options page. Modelos personalizados ainda não são salvos, sendo necessário inseri-los novamente a cada inicialização do navegador.

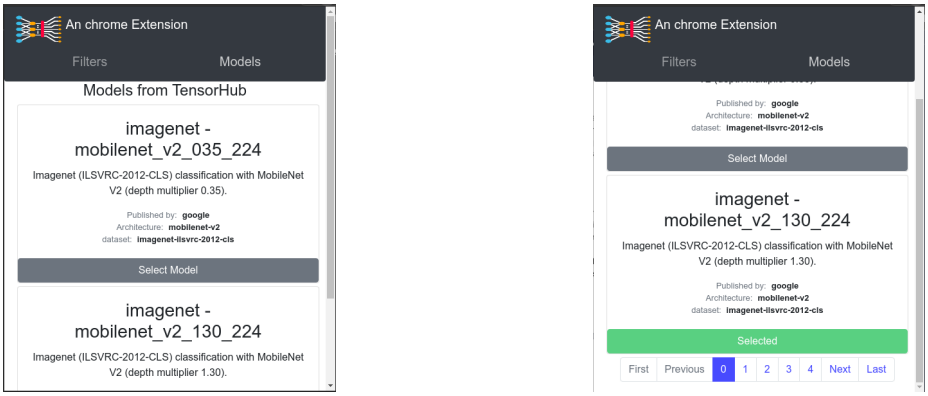


Figura 2: Interface gráfica, lista de modelos, total 45 modelos

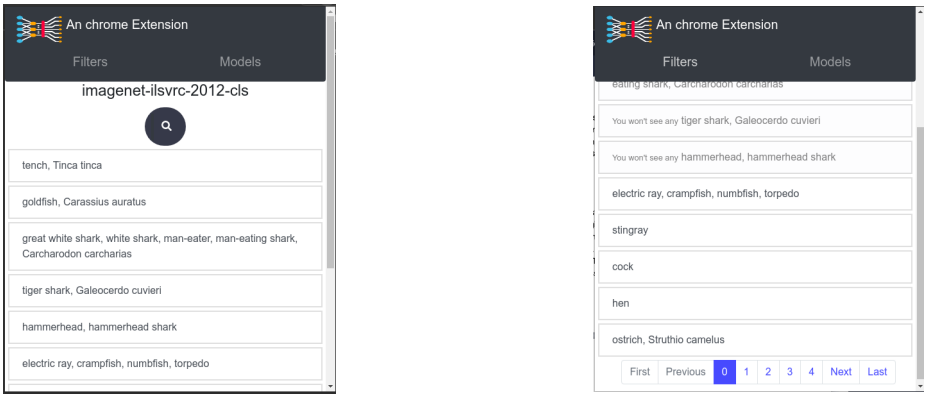


Figura 3: Interface gráfica, Lista de classes para filtragem, total 1000 classes

4.5 Classificador

Essa interface é responsável por receber uma imagem e classificá-la como apresentável ou não. Apesar de uma CNN poder classificar uma imagem em enumeras classes diferentes a decisão final é se essa imagem pertence ao conjunto de imagens que podem ser exibidas ou não. Apos realizar a classificação, ela notifica as outras interface a sua decisão.

4.6 Gerenciador de classificadores

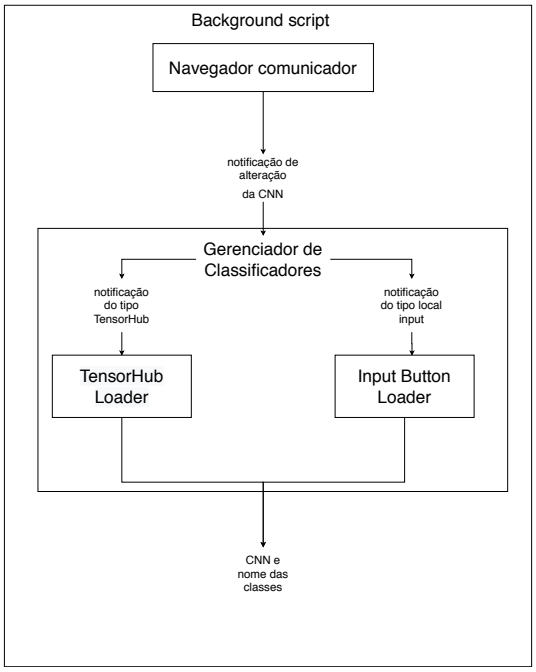


Figura 4: Gerenciador de classificadores

Como surgiu a necessidade de introduzir diferentes formas de selecionar uma CNN, então foi feita essa interface que é responsável por carregar CNN e suas configurações adicionais como todas as classes que ela é capaz de classificar. Internamente existe duas classes, a classe que possui a regra de negocio que carrega as configurações e o modelo do tensorhub e classe que possui a lógica de carregar um modelo e suas configurações localmente. Se surgir a necessidade de adicionar outra origem, é nessa interface que se deve adicionar essa funcionalidade.

4.7 Carregador de imagens

Como não é possível enviar uma imagem diretamente para o background script, optou-se por enviar as URLs das imagens por meio do navegador-comunicador, onde essas URLs são enviadas para o carregador de imagens que instancia um objeto JavasScript do tipo Image, na qual é sobrescarregado o método onload. Assim, quando a imagem for carregada, o carregador de imagens notificar para os outros componentes a imagem instanciada. Em Javascript, o método onload é chamado quando a imagem está carregada. Um fato importante desse método de carregamento é que ele refaz o request da imagem novamente para o servidor e o processo de carregamento da imagem pelo carregador de imagem ocorre em paralelo ao carregamento da imagem na página do usuário, ou seja se por algum motivo o servidor não responder a esse novo request a imagem nunca será exibida. Também é observado que os pedidos de request do background demoram um pouco mais do que na página atual do usuário.

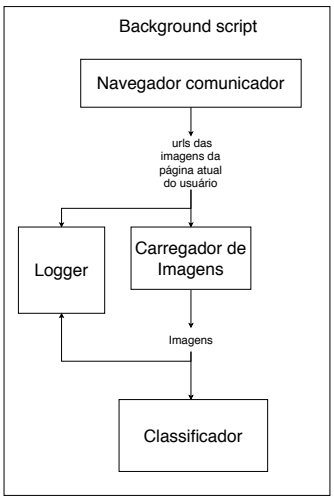


Figura 5: Exemplo de utilização do Logger

4.8 Logger de tempo

Essa interface foi criada com o objetivo de monitorar e armazenar o tempo que determinada interface demora para responder a uma notificação. É através dela que se é capaz de mensurar o desempenho da extensão. O logger monitora as notificações que entram e saem de determinada interface e calcula a diferença de tempo milissegundos entre a notificação que entra e sua respectiva saída, na figura a cima mostra uma exemplo de sua utilização, no caso ele está monitorando a notificação de URLs de imagens que entram e o tempo levado para a interface carregador de imagens leva para carregar uma imagem através da URL, todo logger precisa de um nome único que distingui-los de outros loggers, note que nesse exemplo em questão o logger sabe a diferença de tempo entre a notificação que entra e a que sai pela URL, ou seja o logger anota que a URL entra no carregador de imagens e anota o tempo em que a imagem com a mesma URL sai do carregador de imagens, e então calcula a diferença de tempo.

Como em toda aplicação, só é interessante anotar o tempo quando possui imagens ou URLs de imagens nas notificações, posto que essas URLs foram utilizadas de identificador único. No momento da escrita desse artigo, o logger a partir de 600 valores armazenados, ele exporta os tempos em um arquivo do tipo CSV. Planeja-se no futuro utilizar uma solução como o Cloud Firestore do Firebase, um banco de dados NoSQL para armazenar esses logs.

4.9 Algoritmo de filtragem

Nessa seção será explicada a sequência de passos aplicados pelo sistema para bloquear a visualização de todas as imagens até o momento em que o sistema libera as imagens de acordo com a decisão da CNN. Uma vez que a extensão já está instalada e o usuário entra em uma nova página como por exemplo `facebook.com`, é injetado um arquivo de estilo chamado de hide.css no início de seu carregamento, impossibilitando que qualquer imagem seja exibida. Em seguida, após todo o carregamento do document object model (DOM) ser completado, mas antes que recursos como imagens ou frames sejam completamente carregados, é injetado um código em JavaScript que contém um observador de página a qual notifica para os outros componentes do background script as URLs das imagens observadas através do navegador-comunicador. Uma vez que o background script é

notificado das URLs, ele cria uma nova instância de imagem através da URL e quando a imagem estiver carregada, a imagem é enviada para a CNN onde é classificada, após a CNN tomar usa decisão ela notifica o observador de página para exibir a imagem ou continuar impedindo a sua exibição.

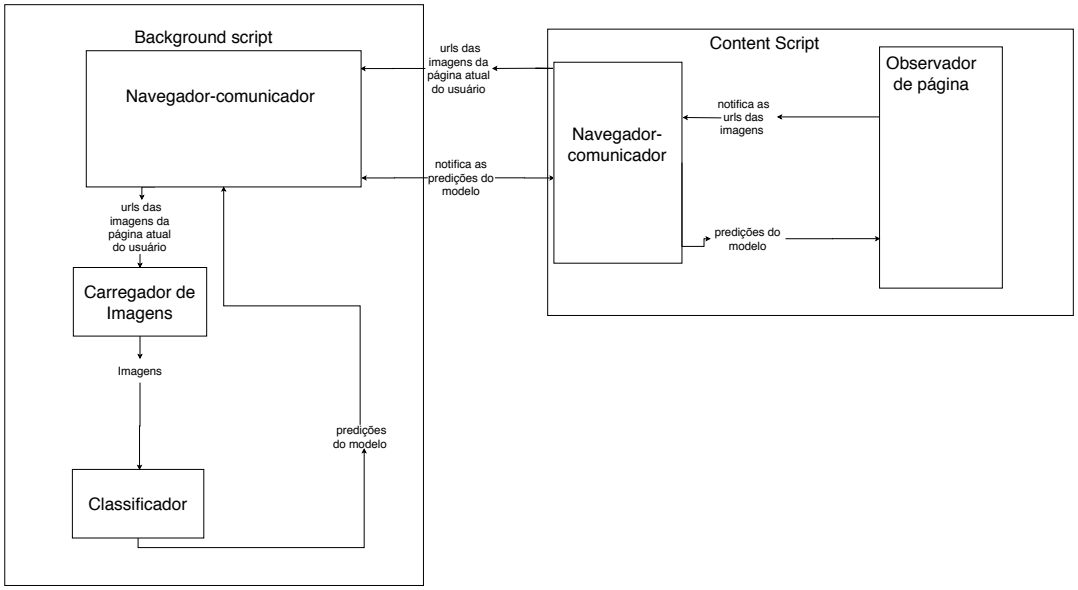


Figura 6: diagrama do algoritmo de filtragem

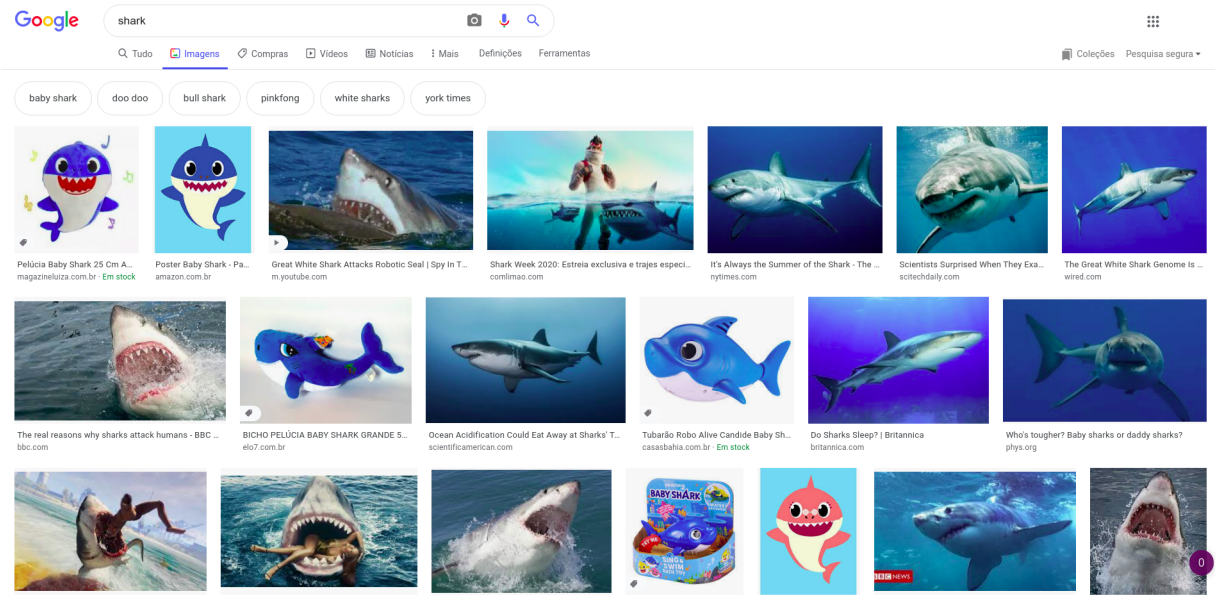


Figura 7: Pesquisa da palavra shark no google imagens sem filtragem

4.10 Carregando uma CNN personalizadada

Além de carregar modelos do tensorhub, essa extensão também tem a capacidade de carregar modelos personalizados desde que siga os seguintes critérios:

- Seja um modelo salvo no formado do framework do tensorflow ou keras.

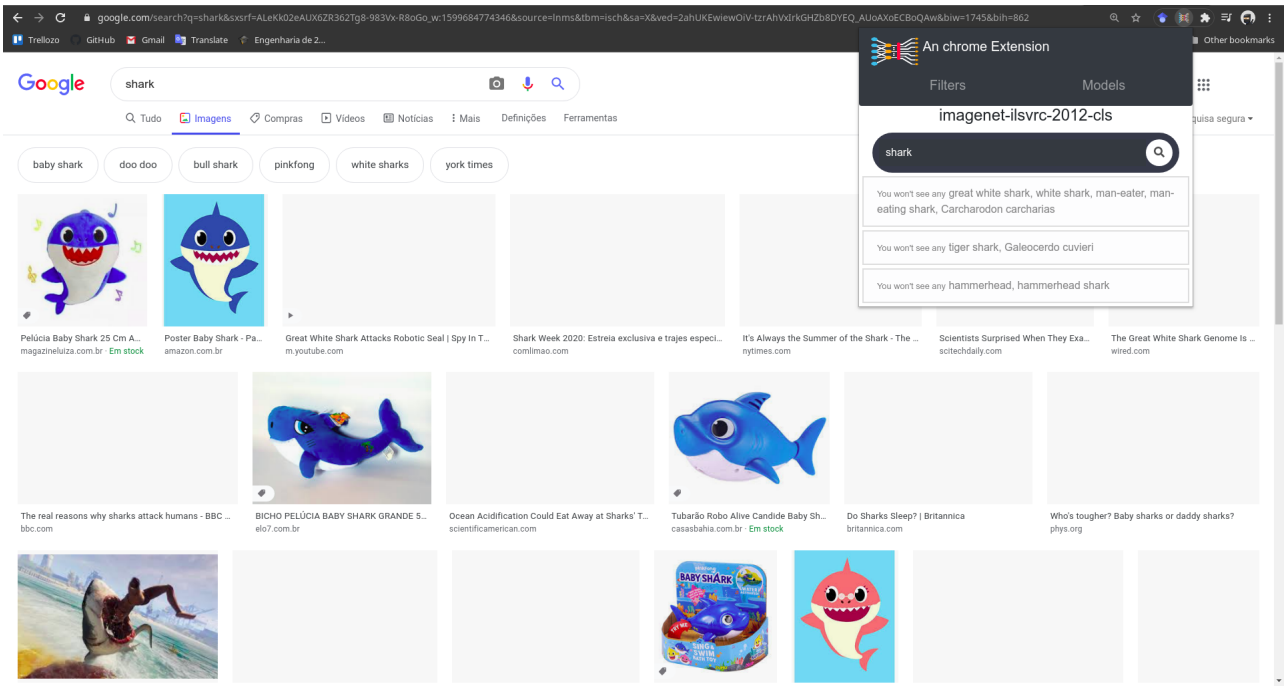


Figura 8: Pesquisa da palavra shark no google imagens filtrando todos os tipos de tubarão conhecidos pelo modelo

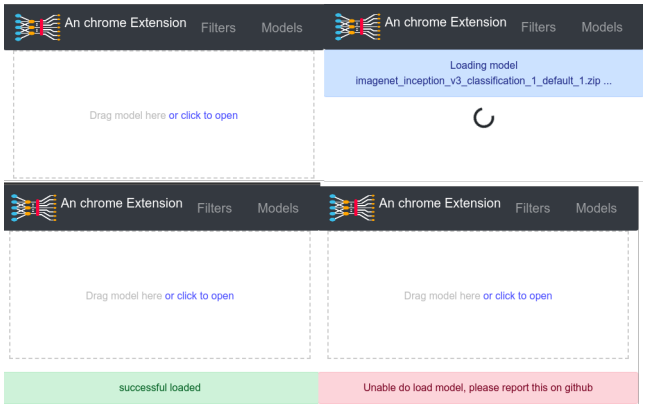
- O modelo deve ser compactado em formado .zip.
- Dentro do arquivo .zip deve existir um outro arquivo chamado settings.json.

O tensorflow oferece uma ferramenta chamada de tensorflowjs_converter essa ferramenta é capaz de transformar um modelos de CNNs feitas em python com a biblioteca de deep learning keras ou com o próprio tensorflow para modelos capazes de serem carregados no tensorflow.js. Para saber mais sobre essa ferramenta deve-se acessar o link `https://www.tensorflow.org/js/tutorials/conversion/import_keras` e, uma vez que tenha a ferramenta instalada, é só usar o commando tensorflowjs_converter para transformar o modelo feito em python em um modelo JavaScript, mas utilizando o formando de saída **tfjs_graph_model** esse formato aumenta a performance da CNN em troca não é possível treinar uma CNN nesse formado. Tendo um modelo de CNN no formato **tfjs_graph_model** do tensorflow.js, adiciona-se a pasta do modelo o arquivo de configuração **settings.json** e configura-se de acordo com a Listagem 5.

Comprime-se a pasta em formato zip e com a extensão instalada deve-se ir na URL `chrome://extensions/` e procurar pela extensão **an chrome extension**, clicar em **details** depois em **Extension options**. Ao aparecer a página da Figura **??**, deposita-se o arquivo .zip na região pontilhada e aguarda-se o carregamento. Se não ocorrerem erros, deverá aparecer uma notificação verde, caso o contrário deverá aparecer uma mensagem vermelha, sendo necessário notificar o erro ao desenvolvedor.

Listagem 5 Exemplo de arquivo settings.json

```
{   "cnnModelHub": {
        "title": "tit lo ou nome do modelo",
        "description": "uma breve descri  o do modelo",
        "publisher": "nome do autor",
        "architecture": "custom", <-- obrigat rio que seja custom
        "dataset": "nome do dataset utilizado, pode ser qualquer um
           ",
        "URL": "Local Data" <-- obrigat rio que seja Local Data
  },
  "classNames":{ <-- classes que a CNN eh capaz de predizer
        "0": "classe 1",
        "1": "classe 2"
    }
}
```



5 Testes e Resultados

Nos testes, foi utilizado um computador dotado de um processador AMD Ryzen 5 2500u com radeon vega mobile gfx x 8, que possui 8 threads, 15.3 Giga de memoria RAM, no sistema operacional utilizado é o Arch linux, kernel versão 5.6.7 com compilação personalizadada para melhor desempenho em processadores AMD. O navegador utilizado foi o Chromium Version 85.0.4183.83. Para a criação dos gráficos foram monitoradas seiscentas classificações da CCNN MobileNet V2 e mais seiscentas classificações da CNN personalizadas. Os gráficos são mostrados nas Figuras 9a, 10b foram montados usando mil e duzentos pontos, e os gráficos 10a, **??**, 9b, seiscentos pontos

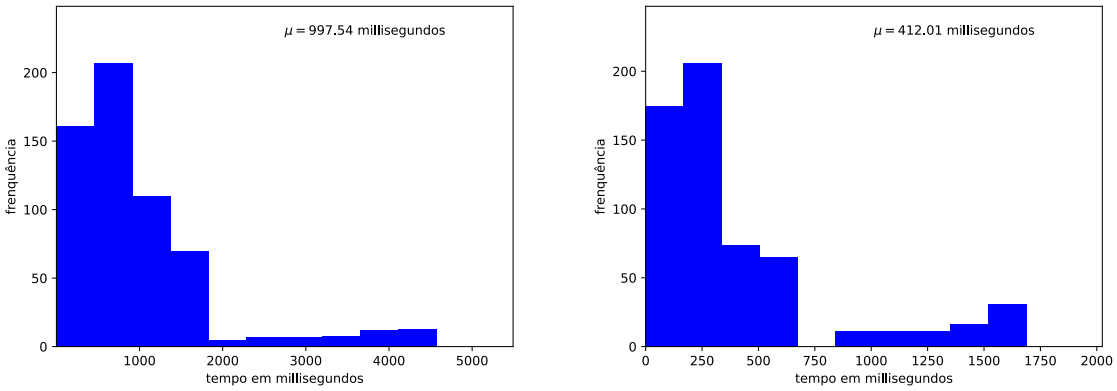
5.1 Descrição do Teste

O objetivo maior dessa extensão é aprimorar a experiência de navegação do usuário, portanto a solução ideal seria tornar imperceptível a queda de performance, pensando nisso foi decidido avaliar o desempenho em relação ao tempo gasto pela extensão para exibir uma imagem e comparar com tempo gasto pelo navegador sem a extensão. Com o intuído de estressar o sistema e mensurar o desempenho da extensão foi criado o seguinte teste:

- Primeiro é habilitado os loggers do sistema, que monitoram o observador de página, a CNN e

o carregamento da imagem no background script.

- Segundo é acessado o google imagens e inicia-se a busca por memes, caso esteja utilizado a rede personalizadada ou busca-se por tubarão caso seja uma das. redes do *https://tfhub.dev/*.
- Terceiro, desça até acabar os memes ou as imagens de de tubarão ou até os arquivos csv do loggers começarem a baixar.
- Quarto, utilizando as ferramentas de desenvolvedor do chrome, desative a memória cache e comece a monitorar os requests, recarregue a página e veja mais memes ou tubarões até acabar os memes ou tubarões.
- Quinto, utilizando as ferramentes de desenvolvedor do chrome exporte o HTTP Archive format do monitoramento filtre os requests de imagem e exporte como csv.
- Sexto, faça os gráficos.



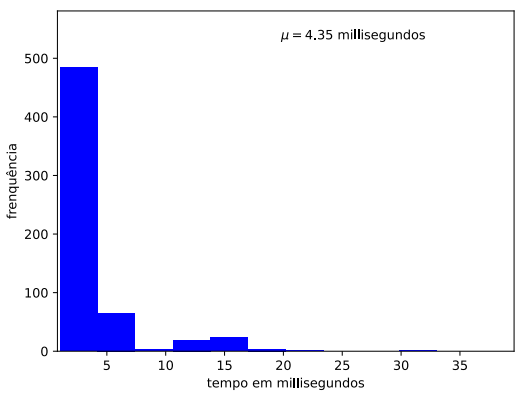
(a) exibir uma imagem com extensão

(b) exibir uma imagem sem extensão

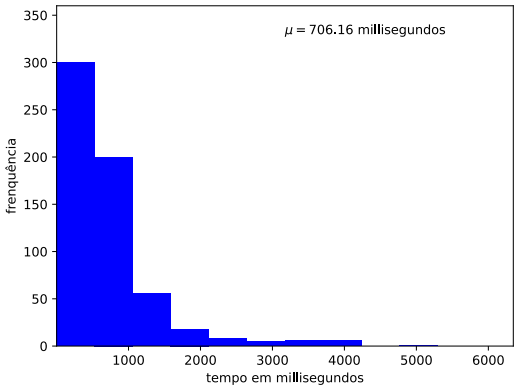
Figura 9: comparação entre tempos de espera para exibir uma imagem de com e sem a extensão, onde μ é a média aritmética das medidas

5.2 Desempenho da extensão

O teste descrito foi feito duas vezes, a primeira utilizando a CNN Inception carregada direto do tensorhub e a segunda usando a mesma CNN mas carregada localmente, O primeiro fato a ser destacado é o baixo tempo de execução das CNN comparando com o tempo que o navegador necessita para carregar as imagens, mostrando também que ela não são o gargalo do sistema, No entanto durante a execução dos testes observou-se travamentos no navegador, pensa-se que esse travamentos ocorrem devido ao alto consumo de CPU, uma vez o tensorflow.js utiliza muito desse recurso, mas durante uma navegação menos estressante, não se observou tais travamentos. Outro fato é que a operação que mais demanda tempo é o carregamento da imagem no background script, uma vez que não se pode transferir o objeto imagem do content script para o background script, optou-se por fazer o recarregamento dessa imagem através da URL. Carregamento que ocorre em paralelo ao pedido pela página principal. Ë importante notar que esses os resultados desses destes de carregamento possuem grande dependência com a velocidade de internet e de qual próximo



(a) Inception versão 1



(b) Tempo para uma imagem ser carregada no background

Figura 10: comparação entre tempos de esperada da CNN Inception predizer uma imagem e a imagem a ser carregada no background, μ é a média aritmética das medidas

está do servidor está a imagem. No entanto em todos os testes observou-se que o background script sempre terá um tempo maior de espera pelo request do que a página principal. Também é importante destacar a exibição da imagem está atrelada ao sucesso do request no background script, mesmo que na aba principal o request teve sucesso, se por algum motivo o background não conseguir se comunicar com o servidor a imagem não será mostrada, no entanto esse fato não foi observado nos testes.

6 Conclusão

O presente trabalho apresentou uma extensão para navegadores baseados em Google Chrome capaz de filtrar quaisquer tipos de imagens baseadas no reconhecimento de características via redes de aprendizado profundo. A extensão é configurável, permitindo ao seu utilizador carregar uma estrutura de rede que será usada para reconhecer as imagems em tempo de execução.

Experimentos realizados com bancos de dados de repositórios públicos mostraram que a extensão possui capacidades promissoras de filtrar conteúdo, sendo dependente apenas da qualidade da rede que é pré-treinada para realizar a classificação. Verificou-se que, embora tenha havido retardo na exibição das imagens conforme o filtro que é configurado, o retado pode ser aceitável se o processo de filtragem é mandatório para o usuário. Por exemplo, se o equipamento destina-se ao uso de uma criança, carregar um bom filtro de classificação e bloqueio de imagens adultas poderá deixar a experiência de uso mais lenta, porém justificável pelo esforço de não exibir conteúdos sensíveis por acidente.

A ferramenta ainda carece de testes e aprimoramentos, bem como melhorias de usabilidade, mas os esforços voltados para o seu desenvolvimento são válidos, posto que extensões dessa natureza ainda não oferecem os recursos inovadores que foram aqui apresentados.

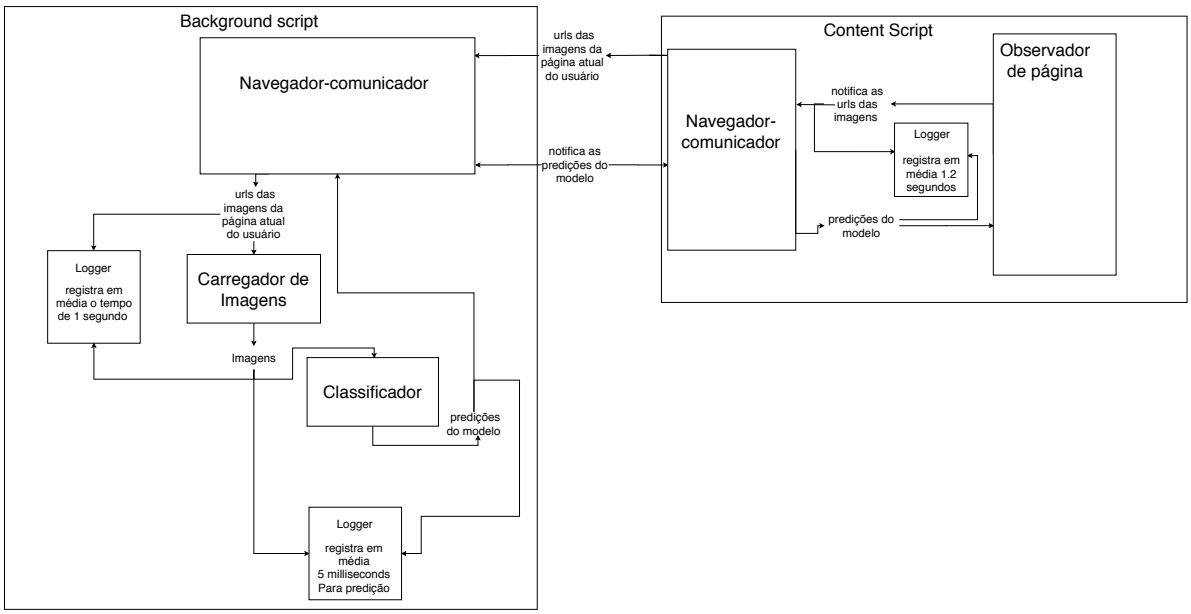


Figura 11: diagrama do algoritmo de filtragem com loggers

Referências

[1] Google, *Extensions: maximize the Chrome browsing experience*, 2020 (acesso em Setembro de 2020), https://developer.chrome.com/home.

[2] A. C. G. Vargas, A. Paes, and C. N. Vasconcelos, "Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres," in *Proceedings of the xxix conference on graphics, patterns and images*, vol. 1, no. 4, 2016.

[3] S. K. Kasagoni, *Building Modern Web Applications Using Angular*. Packt Publishing Ltd, 2017.

[4] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen, D. Soergel *et al.*, "Tensorflow. js: Machine learning for the web and beyond," *arXiv preprint arXiv:1901.05350*, 2019.

[5] D. Flanagan and G. M. Novak, "Java-script: The definitive guide," 1998.