Redes neurais artificias de K-Nearest Neighbors para ensinar libras com ml5js

Caroline Pacheco da Rosa <carol@rede.ulbra.br>
Maria Adelina Raupp Sganzerla <maria.sganzerla@ulbra.br>
Universidade Luterana do Brasil (Ulbra) – Curso de Ciência da Computação – Campus Gravataí Av. Itacolomi, 3.600 – Bairro São Vicente – CEP 94170-240 – Gravataí - RS
21 de maio de 2020

Objetivos

O objetivo principal desta rede neural é a de ser um classificador de imagens de sinais usando o algoritmo K-Nearest Neighbors.

O escopo de resultados desejados foi reduzido pelo pouco tempo de implementação determinado. Anteriormente sendo algumas palavras, alfabeto e números, agora serão apenas as letras a, b, f e s, mas que fosse possível, futuramente, aumentar o número de letras. Sendo apresentado para o usuário o sinal que a rede neural conseguiu detectar com base nos exemplos já classificados e no próprio sinal mostrado pelo usuário a câmera.

Além disso, é desejável que a aplicação seja de fácil utilização e acesso, por isso a escolha de usar ferramentas para navegadores web.

Descrição da RNA

O algoritmo K-Nearest Neighbors é usado para criar um classificador único de qualquer dado que possa ser classificado. Diferente de outras funções de classificação de imagem do ml5js (framework de machine learning em JavaScript), é possível criar um modelo KNN para classificar as imagens, recebidas a partir do vídeo por uma webcam, em quaisquer teclas do teclado, sendo possível a classificação dos sinais de letras em linguagens de sinais.

Ele não precisa de um modelo pré-treinado, mas sim um utilitário para construir outros exemplos. Na utilização desta rede neural neste trabalho, utiliza-se a função para extrair a imagem do usuário e treinar a rede neural para entender as imagens (sinais em libras) que são gravadas pelo vídeo. Cada imagem alimenta o KNNClassifier, ou classificador, que é descrito para a rede neural.

Por exemplo, se temos um sinal no vídeo da letra a, o classificador entenderá que esta é uma imagem da tecla a. Veja que o classificador não tem nenhuma outra imagem em seu conhecimento, logo não irá supor que seja outro sinal. Adicionando a imagem do sinal b, temos então a impressão de que agora o classificador está decidindo e comparando entre as duas imagens de a e b, e apresentando para o usuário seu resultado.

Desenvolvimento RNA

Muito do que foi implementado na aplicação são funções prontas do ml5js e do p5js, ambas bibliotecas de JavaScript. ml5js é uma biblioteca em javascript que utiliza p5js para criar redes neurais e outras aplicações de machine learning (aprendizado de máquina). p5js é também uma biblioteca em JavaScript, muito intuitiva, usada basicamente para desenho em tela. Ambas ferramentas são de código aberto e estão em desenvolvimento constante.

Em questões de ambiente, a aplicação está sendo servida localmente por um servidor em nodejs, framework para interpretar JavaScript fora do cliente web, além de manipular conexões ao site. A aplicação tem alto acesso em quase qualquer navegador, um dos objetivos deste projeto, desde que o usuário permita a utilização de sua câmera.

Toda aplicação p5js começa com uma função principal setup() na qual a mesma cria a tela onde serão dispostos os ambientes de vídeo e visualização da câmera, cria-se o objeto ml5 que utiliza o modelo MobileNet para extrair a imagem e os métodos de classificação e treinamento que virão a seguir.

Treinamento

A função key() do p5js é utilizada para adição de exemplos de imagens com as palavras, ao clicar em uma tecla, por exemplo “a”, a imagem do usuário é relacionada com a tecla pressionada, essa classificação é salva em um arquivo json. Ao mesmo tempo em que estamos salvando essas imagens, os dados coletados alimentam o KNNClassifier com a função infer() classificando as imagens e imprimindo na tela a solução encontrada pela rede neural a partir da classificação, ou seja, basicamente apresentará para o usuário qual sinal está sendo “visualizado” pela rede neural.

Pode-se colocar qualquer sinal sobre qualquer tecla do teclado, mas espera-se que o usuário que está treinando a rede tenha consciência de enviar os mesmos sinais com as mesmas teclas para que a rede não seja treinada incorretamente.

Resultados

Após o treinamento a aplicação pode ser alterada para uma versão de teste. É utilizado nesta versão o treinamento salvo no arquivo json.

Apesar da ferramenta ser de fácil utilização e implementação, a maior dificuldade deste projeto está no envio de imagens (dados) à rede neural. Adiciona-se uma letra e não terá outros resultados que não seja esta letra. Por isso, optou-se por adicionar uma tecla curinga para inválido ou “não tem letras”, isto é, adiciona-se várias imagens onde o usuário não está apresentando para a câmera algum sinal a fim de validar se está ou não correto o sinal do usuário para letras.

Neste cenário, com 2 exemplos (uma letra e exemplo curinga) precisa-se de pelo menos 2 imagens de cada para que a rede neural seja bem treinada. Pensa-se que os exemplos para mais letras precisam então serem multiplicados por 2, exemplo: 2 letras com um exemplo curinga, 6 imagens de cada. Entretanto, o número de exemplos necessários para um treinamento e classificação mínima é exponencial. Conforme teste, adicionar mais uma letra no cenário atual (2 letras e um exemplo curinga) será necessário pelo menos 20 imagens do sinal em diferentes posições da tela, temos então 60 imagens num cenário de 2 letras e um exemplo curinga. Com 4 sinais e um exemplo curinga, precisamos de, aproximadamente, 250 imagens, resultando em um baixo treinamento. Por esse motivo foi delimitado um espaço na tela do usuário para que o mesmo mostre o sinal para a câmera na posição correta, afim de não aumentar a quantidade de imagens necessárias.

Ainda há motivos para termos resultados favoráveis. Apesar do projeto precisar de melhorias de UX - em pequeno teste com a aplicação, alguns usuários tiveram dificuldade para assimilar onde deveriam mostrar o sinal, não identificando a área marcada, e não conseguiram ter um retorno correto do sinal que mostravam à câmera -, em teste com alguns usuários, a aplicação conseguiu determinar corretamente o sinal que era mostrado, comprovando que falta explicações ou um vídeo explicativo ou práticas corretas de IHC.

Em questões de implementação, ao acessar o arquivo json com os treinamentos salvos, o ml5js apresenta um bug de desenvolvimento, já conhecido pela sua comunidade. A função que abre este arquivo e visualiza os resultados, acaba por não trazer corretamente as informações de exemplos do json. Ao invés de trazer o resultado real, mostra o índice onde está o resultado dentro dos objetos que são criados no json. Foi necessário uma adaptação do código para que o usuário recebesse o resultado correto.

Considerações finais

Com certeza esse projeto, pelo seu tempo curto de implementação, precisa de aprimoramentos e maiores treinamentos. A aplicação demonstrou resultados favoráveis e é de extrema importância que a mesma seja usada com usuários reais. Usar com usuário em treinamento de linguagem de sinais ou não ouvintes, para validar os resultados. Além de necessitar-se maiores pareceres sobre a utilização da mesma, adaptando às necessidades reais dos usuários.

Existem poucas aplicações acessíveis, ou poucas que sejam acessíveis em português, para que o usuários aprendam linguagens de sinais através do seu navegador, sem que seja instalado algum software auxiliar ou aplicativo. Ademais, é interessante que seja uma aplicação que corrija sinais incorretos e determine se o usuário está aprendendo a linguagem. Com isso, conclui-se que o projeto merece espaço e dedicação futura para suprir as atuais dificuldades e para que seja verdadeiramente usado para o aprendizado.

Referências

ML5 Beginners Guide. The Coding Train, 2020. Disponível em: https://thecodingtrain.com/learning/ml5/ Acesso em: 21 de maio de 2020.

STUDY GUIDE. Life Print, 2020. Disponível em: http://www.lifeprint.com/asl101/pages-signs/. Acesso em: 21 de maio de 2020.

KNNClassifier: K-Nearest Neighbors Classifier. ml5js, 2020. Disponível em: https://learn.ml5js.org/docs/#/reference/knn-classifier. Acesso em: 21 de maio de 2020