PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Inteligência Artificial e Aprendizado de Máquina

RODRIGO MARQUES PESSOA

VISÃO COMPUTACIONAL NO RECONHECIMENTO DE PLACAS DE VEÍCULOS COM REDES NEURAIS CONVOLUCIONAIS

Campinas
2021

RODRIGO MARQUES PESSOA

VISÃO COMPUTACIONAL NO RECONHECIMENTO DE PLACAS DE VEÍCULOS COM REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Inteligência Artificial e Aprendizado de Máquina como requisito parcial à obtenção do título de especialista.

Campinas

2021

SUMÁRIO

1. Introdução

1.1. Contextualização

1.1.1 Visão Computacional

A visão computacional é uma área de atuação do campo da inteligência artificial. Essa visão tem transformado a realidade de diversos setores da sociedade. O seu objetivo é replicar funções da visão humana. Isso é possível por conta do uso de softwares e hardwares avançados.

Muitas pessoas têm dificuldade em imaginar que máquinas são realmente capazes de replicar capacidades biológicas. No entanto, tal tecnologia já é real e capaz de realizar diversas tarefas, como detecção, restauração de imagens, reconhecimento, movimento e identificação, por exemplo.

Dessa maneira, diversas aplicações têm aparecido em diferentes áreas de atuação, sendo bastante úteis para o desenvolvimento tecnológico e operacional.

Para que não haja um número alto de erros, o sistema é alimentado com milhares de imagens relacionadas a determinado assunto. Assim, fica muito fácil para o computador realizar correlações entre objetos específicos.

Por exemplo, ao invés de fazer com que o computador procure por patas, orelhas e narizes com o objetivo de identificar um gato, faz-se o upload de variadas fotos de gatos. Com isso, automaticamente o sistema aprende sozinho quais são as possíveis características que esse animal pode apresentar, a fim de identificar mais facilmente um gato.

Por ser uma tecnologia inovadora e altamente eficaz, a visão computacional está sendo bastante usada em diversas áreas, a fim de reduzir custos, otimizar processos, melhorar a experiência do cliente e aumentar a segurança, por exemplo.

Assim, as principais tarefas em que a visão computacional é usada são:

- Reconhecimento Facial de Emoções;
- Reconhecimento e Detecção de Atividade Humana;

- Detecção de Veículos;
- Processamento e Geração de Vídeos.

Tal tecnologia supera as capacidades humanas em muitas áreas, como a análise de exames médicos, como raios-x, tomografias e ressonâncias magnéticas com o objetivo de verificar a existência de alguma anomalia, por exemplo.

1.1.2 Redes Neurais Convolucionais(CNN)

Uma Rede Neural Convolucional (ConvNet / Convolutional Neural Network / CNN) é um algoritmo de aprendizado profundo que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos da imagem e ser capaz de diferenciar um do outro. O pré-processamento exigido em uma ConvNet é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são feitos à mão, com treinamento suficiente, as ConvNets têm a capacidade de aprender essas características.

Uma CNN pode ser dividida em duas partes: extração de características (*Conv*, *Padding, Relu*, *Pooling*) e uma rede neural tradicional.

1.2. Convoluções

Matematicamente, uma convolução é uma operação linear que a partir de duas funções, gera uma terceira (normalmente chamada de *feature map*). No contexto de imagens, podemos entender esse processo como um filtro/*kernel* que transforma uma imagem de entrada.

Um *kernel* é uma *matrix* utilizada para uma operação de multiplicação de matrizes. Esta operação é aplicada diversas vezes em diferentes regiões da imagem. A cada aplicação, a região é alterada por um parâmetro conhecido como *stride.*
Normalmente o stride possui o valor 1, o que significa que a transformação será aplicada em todos os *pixels* da imagem

1.3. ReLU

Uma rede neural sem função de ativação torna-se um modelo linear. Se o seu problema é linear, existem outros modelos mais simples que te atenderão tão bem quanto uma rede neural. Infelizmente a maioria dos problemas complexos não são lineares. Portanto, para adicionar a não linearidade a rede, utilizamos as funções de ativação. Nos dias de hoje, e principalmente no contexto de imagens, a mais utilizada é a função ReLU.

1.4. Pooling

É um processo simples de redução da dimensionalidade/*features maps*. Em uma forma leviana de pensar, podemos entender essa transformação como uma redução do tamanho da imagem.

A principal motivação dessa operação no modelo, é de diminuir sua variância a pequenas alterações e também de reduzir a quantidade de parâmetros treinados pela rede.

1.5. Flatten

Essa camada normalmente é utilizada na divisão das 2 partes da CNN (extração de características / rede neural tradicional). Ela basicamente opera uma transformação na *matrix* da imagem, alterando seu formato para um array. Por exemplo, uma imagem em *grayscale* de 28x28 será transformada para um array de 784 posições.

1.6. Rede Tradicional (Dense Layers)

Rede neural é um modelo computacional baseado no sistema nervoso central humano. Elas são capazes de reconhecer padrões em uma massa de dados de forma a classificá-los em alguma categoria ou fazer a regressão de algum valor.

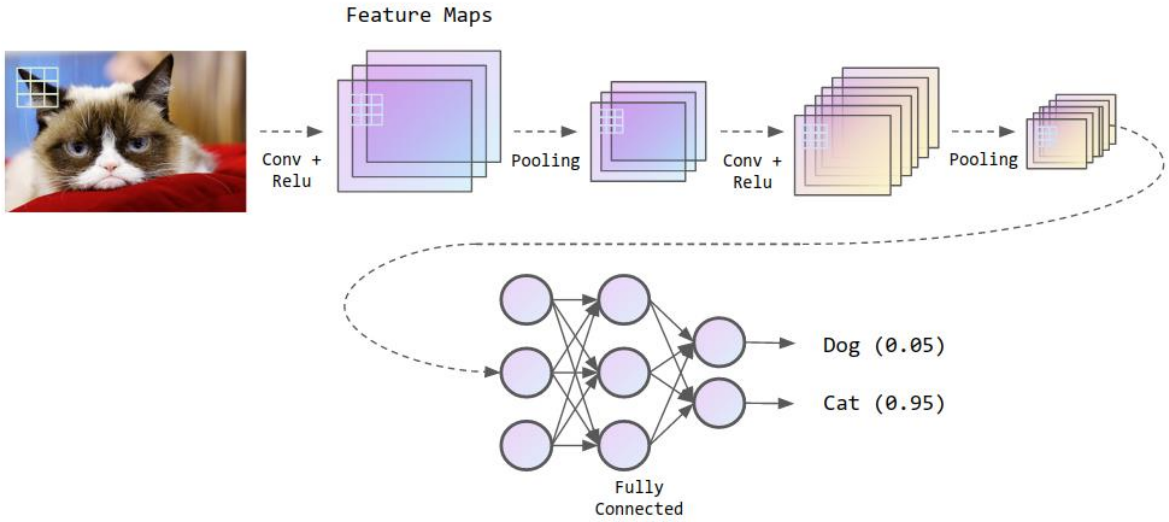

Imagem - Arquitetura CNN.

1.7. O problema proposto

Para os estabelecimentos comerciais é muito importante identificar a origem dos seus clientes para que possam fazer uma campanha de marketing direcionada para uma determinada região ou cidade.

Pensando nisso será criado um processo para identificar a origem das placas. Nas placas antigas essa informação pode ser obtida diretamente nas placas, mas nas novas placas essa informação não é mais apresentada. O sistema de identificação irá coletar a imagem da placa, realizar o processo de identificação das letras e números da placa através da utilização de uma rede CNN pré-treinada e realizar uma pesquisa na internet para identificar o estado e a cidade da placa e salvar essa informação em uma base de dados para posterior utilização em dashboards para que o departamento de marketing possa criar campanhas direcionadas para estes estados ou cidades.

2. Coleta das Imagens

A coleta das imagens que será utilizado para treinar a CNN será feita manualmente através do mecanismo de busca de imagens do Google. As imagens são as placas de veículos que terão cada letra recortada e etiquetada no nome do arquivo.

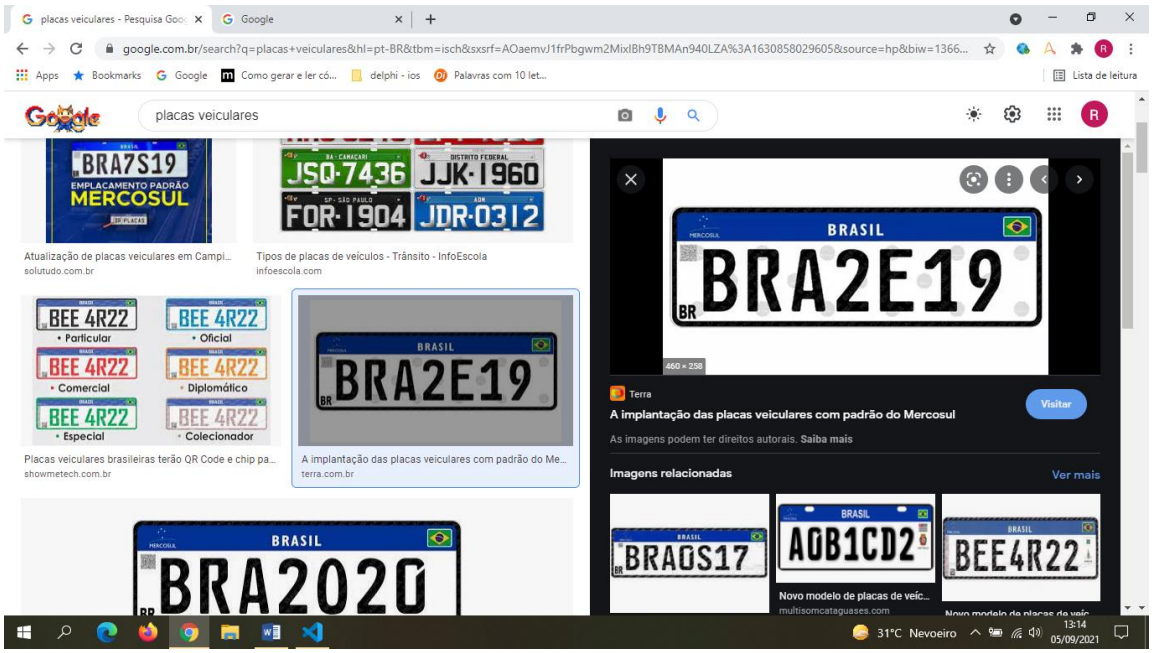

Imagem - Placas coletadas da internet.

Após a etiquetação das imagens das placas teremos os números e letras etiquetadas pelo nome da imagem que servirão de imagens de treinamento para CNN.

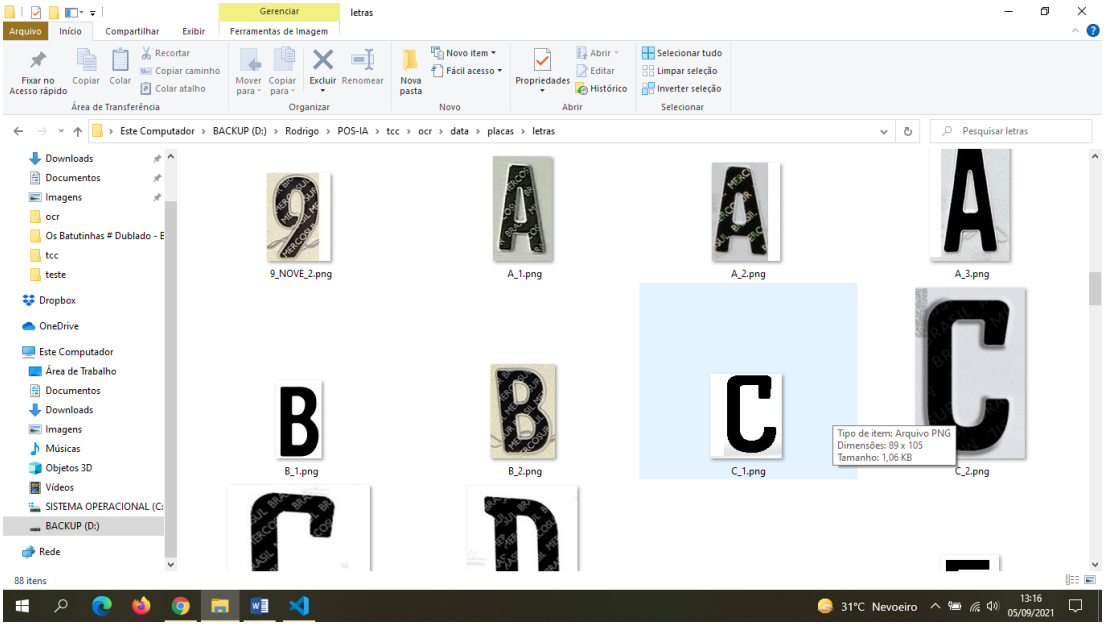

Imagem - Caracteres das placas etiquetados.

3. Processamento/Tratamento de Dados

A etapa de processamento e tratamento consiste em atuar sobre os dados obtidos para que eles possam ser utilizados no treinamento da CNN. Nesta etapa as imagens serão dimensionadas para um tamanho padrão e convertidas para escala de cinza para facilitar o processamento já que nesses casos os canais de cores não são necessários.

Para aumentar o número de imagens será aplicada também técnicas de erosão e dilatação e por fim rotação gerando novas imagens que ajudarão a rede neural na identificação das imagens das placas. Para realizar estas tarefas utilizaremos a biblioteca do open CV conforme código abaixo:

```
def preprocessamento(pathin, pathout, maxsize):
    tam = len(pathin.split('\\'))
    filename = pathin.split('\\')[tam-1]

    img = cv2.imread(pathin)
    img_resize = cv2.resize(img,maxsize,cv2.INTER_AREA)
    gray = cv2.cvtColor(img_resize, cv2.COLOR_BGR2GRAY)

    val, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRE
SH_OTSU)
    cv2.imwrite(pathout + '\\' + filename , thresh)

    kernel = np.ones((1, 1), np.uint8)

    img_erosao = cv2.erode(thresh, kernel)
    cv2.imwrite(pathout + '\\' +  '_ero_' + filename , img_erosao)

    img_dilata = cv2.dilate(thresh, kernel)
    cv2.imwrite(pathout + '\\' +  '_dil_' + filename , img_dilata)

    img_rotatel = rotate_image(thresh,5)
    cv2.imwrite(pathout + '\\' +  '_rot_l_' + filename , img_rotatel)

    img_rotater = rotate_image(thresh,-5)
    cv2.imwrite(pathout + '\\' +  '_rot_r_' + filename , img_rotater)

    return (thresh,img_rotatel,img_rotater,img_erosao, img_dilata)
```

Código - Pré-processamento das imagens utilizadas para treinamento da CNN.

4. Criação da Rede Neural Convolucional (CNN)

4.1.Arquitetura

Utilizaremos as bibliotecas do Keras para criar a arquitetura da CNN, contendo uma camada de convolução para extrair as características da imagem seguida por uma camada de maxpolling para reduzir a dimensionalidade e retornar os aspectos mais importantes da imagem. Esse processo será aplicado mais duas vezes. O resultado dessa matrix será convertido para um array (Flatten) que será a entrada da camada de uma rede totalmente conectada que fara o cálculo dos pesos e por fim uma camada que fara a previsão utilizando a função softmax. O código pode ser visto abaixo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense
from tensorflow.keras.callbacks import ModelCheckpoint
rede_neural = Sequential()

rede_neural.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', input
_shape=(28,28,1)))
rede_neural.add(MaxPool2D(pool_size=(2,2)))

rede_neural.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', paddi
ng='same'))
rede_neural.add(MaxPool2D(pool_size=(2,2)))

rede_neural.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu', padd
ing='valid'))
rede_neural.add(MaxPool2D(pool_size=(2,2)))

rede_neural.add(Flatten())

rede_neural.add(Dense(64, activation='relu'))
rede_neural.add(Dense(128, activation='relu'))

rede_neural.add(Dense(36, activation='softmax'))

rede_neural.compile(loss = 'categorical_crossentropy', optimizer = 'adam', met
rics = ['accuracy'])
rede_neural.summary()
```

Código – Definição da arquitetura CNN.

4.2.Treinamento

Os dados serão divididos em treino e teste e será realizado o treinamento da rede e o resultado do treinamento será salvo para posterior utilização no processo de previsão dos caracteres das placas.

```
epochs = 40
batch_size = 20
checkpointer = ModelCheckpoint(arquivo_modelo, monitor='val_loss', verbose=1,
save_best_only=True)
history = rede_neural.fit(augmentation.flow(X_train, y_train,
      batch_size=batch_size), validation_data = (X_test, y_test),
      steps_per_epoch = len(X_train) // batch_size,
      epochs = epochs,
      class_weight = classes_peso,
       verbose=1, callbacks=[checkpointer])
previsoes = rede_neural.predict(X_test, batch_size=batch_size)
print(previsoes)

rede_neural.save('rede_neural_placa', save_format='h5')
```

Código – Treinamento da CNN.

Foi definido que a rede fará o treinamento em 40 épocas pois é o momento que ocorre praticamente o ponto de máxima acurácia entre os dados de treino e validação e se estabiliza nas épocas seguintes conforme mostra o gráfico abaixo.

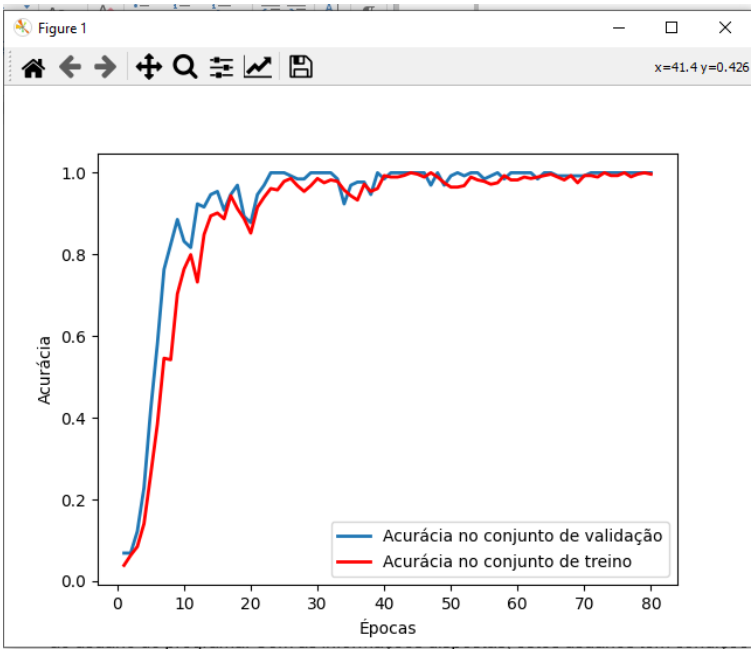

Imagem – Evolução do treinamento.

5. Apresentação dos Resultados

5.1.Identificando a placa

Uma imagem da placa será capturada no momento que o veículo chegar através de uma câmera posicionada na entrada e será enviada para processamento no servidor. Essa imagem será dimensionada e será aplicado uma técnica para extração das bordas para que cada caractere da placa possa ser recortado e posteriormente enviados para a CNN para realizar a predição dos caracteres. Segue um passo a passo do processo e da imagem da placa sendo identificada e o respectivo código fonte.

5.1.1 Passo 1: Captura da Imagem

```
imgcar = cv2.imread(imagemcar)
cv2_imshow(imgcar)

originalcar = imgcar.copy()

H = imgcar.shape[0]
W = imgcar.shape[1]
print(H, W)

proporcao_W = W / float(largura)
proporcao_H = H / float(altura)
print(proporcao_W, proporcao_H)

imgcar = cv2.resize(imgcar, (largura, altura))
H = imgcar.shape[0]
W = imgcar.shape[1]
print(H, W)

cv2_imshow(imgcar)
```

Código - Leitura da placa capturada.

Imagem - Placa capturada.

5.1.2 Passo 2: Tratamento da imagem para capturar bordas

```
if img.shape[1] > tm_placa:
        gray, img_processada = preprocessar_img(img)
        cv2_imshow(gray)
        cv2_imshow(img_processada)

        conts = encontrar_contornos(img_processada.copy())
        caracteres = []
        for c in conts:
            (x, y, w, h) = cv2.boundingRect(c)
            if (w >= l_min and w <= l_max) and (h >= a_min and h <= a_max):
                processa_caixa(gray, x, y, w, h)

        caixas = [b[1] for b in caracteres]
        caracteres = np.array([c[0] for c in caracteres], dtype="float32")
```

Código – Pré-processamento da imagem da placa.

Imagem - Extração das bordas dos caracteres da placa veicular.

Imagem - Identificação dos caracteres para envio individual a CNN

5.1.3 Passo 3: Realizar a predição com a CNN

```
caracteres = np.array([c[0] for c in caracteres], dtype="float32")
previsoes = rede_neural_carregada.predict(caracteres)

placa = ''
img_cp = img.copy()
for (previsoes, (x, y, w, h)) in zip(previsoes, caixas):
     i, probabilidade, caractere = predicao(previsoes, lista_caracteres)
     desenhar_na_img(img_cp, caractere)
     placa += caractere

cv2_imshow(img_cp)
```

Código – Predição dos caracteres da placa.

5.1.4 Passo 4: Pesquisa na Web

```
def Buscaplaca(pplaca):

  #browser = webdriver.Chrome(executable_path=r'D:\Rodrigo\POS-
IA\tcc\ocr\web\chromedriver.exe')
  browser = webdriver.Firefox(executable_path=r'D:\Rodrigo\POS-
IA\tcc\ocr\web\geckodriver.exe') # inicia uma instancia
  browser.get('https://www.qualveiculo.net/') # a instancia .get serve para di
recionar um
  browser.maximize_window() # a instancia .maxime_window() d

  text_area = browser.find_element_by_id("placa")
  text_area.send_keys(pplaca)

  btn = browser.find_element_by_xpath('//*[@id="consultar"]') # Já vamos dele
  btn.click() # iniciando a função de click armazenada no btn

  searchTxt=''
  while not searchTxt:
      try:
        searchTxt=browser.find_element_by_id('resultado').text
      except:continue

  print('Texto Pesquisado=' ,searchTxt)
```

Código Pesquisa dados da placa na internet.

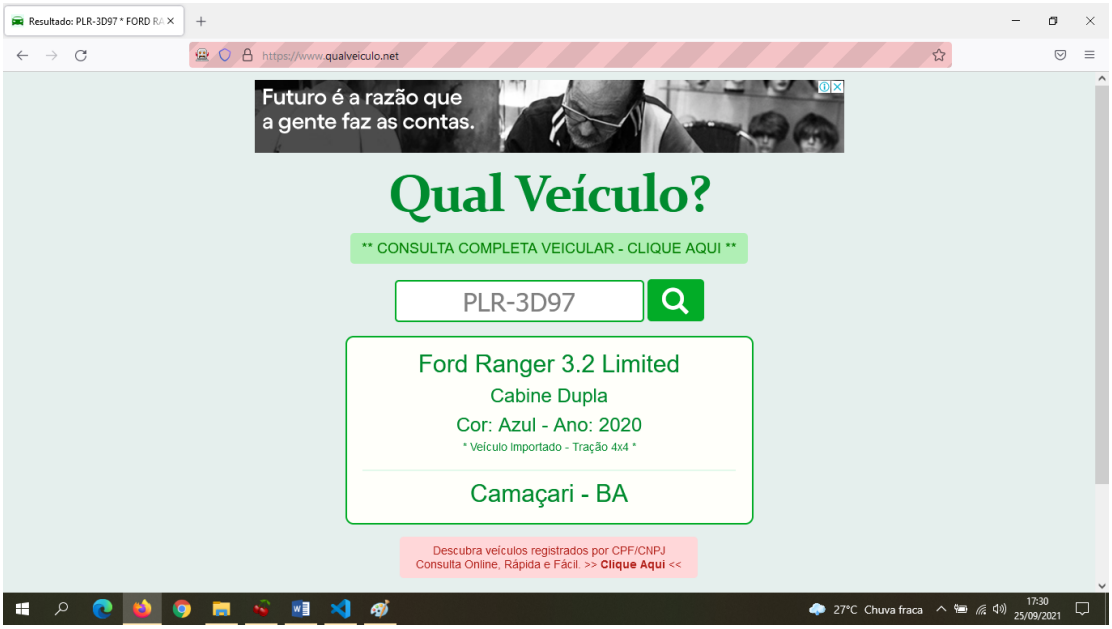

Imagem – Resultado da pesquisa dos dados da placa na internet

5.1.5 Passo 5 – Salvando os dados da placa

```
def SalvaPlaca(pplaca, pestado, pcidade):

  datetime_now = datetime.now()

  try:

    myclient = pymongo.MongoClient("mongodb://localhost:27017/")
    mydb = myclient["dbplacas"]
    mycol = mydb["placas"]

    mydict = {"Placa": pplaca, "Estado": pestado, "Cidade": pcidade, "Data": d
atetime_now}
    id_placa = mycol.insert_one(mydict)

    print("Dados inserido com id",id_placa)

  except:

      print("Erro ao conectar ao MongoDB")
```

Código – Salvando os dados da placa na base de dados.

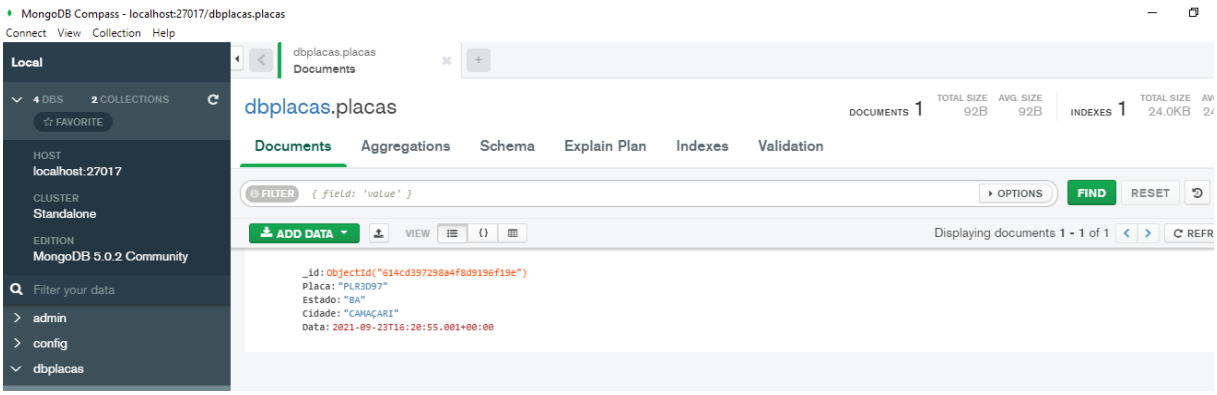

Imagem - Dados da placa salvos em banco de dados MongoDB.

5.1.6 Passo 6 – Apresentando o dashboard com os dados das cidades

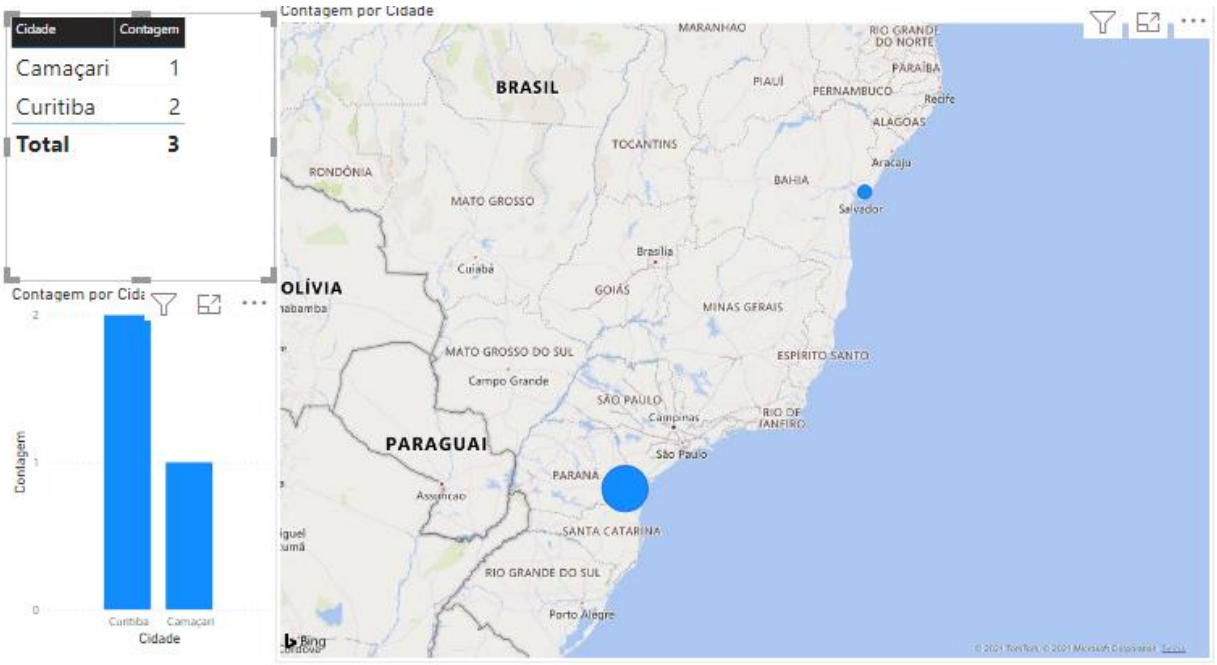

Imagem – Dashboard.

6. Conclusão

Este trabalho tem o objetivo de ser uma prova de conceito e o método proposto para leitura dos caracteres das placas veiculares possui recursos úteis e modernos no campo de inteligência artificial. A utilização da rede neural convolucional (CNN) utilizada permite extrair a informação da imagem da placa e a partir dessa informação realizar a consulta na web para retornar os dados da placa. Sem os artifícios propostos, demandariam tempo de uma pessoa para coletar e pesquisar o que traria custos muito altos principalmente para pequenos negócios.

Conclui-se, portanto, que a técnica é eficaz e atingiu os objetivos propostos.

7. Referências

HIJAZI, Samer; KUMAR, Rishi; ROWEN, Chris. **Using convolutional neural networks for image recognition**. Tech. Rep., 2015. [Online]. Disponível em: <http://ip.cadence.com/uploads/901/cnn-wp-pdf>.

KARPATHY, A. CS231n **Convolutional Neural Networks for Visual Recognition**. 2016. Disponível em: <http://cs231n.github.io/>,

JONES, M. T. **Arquiteturas de aprendizado profundo - o surgimento da inteligência artificial.** IBM developer works - Disponível em: <https://www.ibm.com/developerworks/br/library/cc-machine-learning-deep-learningarchitectures/index.html>.

TensorFlow. TensorFlow — an **Open Source Software Library for Machine Intelligence**.