Real-Time Traffic Sign Detection and Recognition Using CNN

D. Santos, F. Silva, D. Pereira, L. Almeida, A. Artero, M. Piteri, and V. de Albuquerque

***Abstract*—Traffic signs presents on streets and highways have a distinct set of features which may be used to differentiate each one from each other. We propose in this paper a real-time traffic sign detection and recognition algorithm using neural networks. In order to detect traffic sign we used a Faster R-CNN (Region-Based Convolutional Neural Network), and to classify we used a Convolutional Neural Network using two different architectures. Some factors can make it difficult, such as light, occlusion, blurring, and others. This work can be applied in several areas, such as Advanced Driving Assistant System and autonomous cars.**

***Index Terms*—Computer Vision, Convolutional Neural Network, Region-Based Convolutional Neural Network.**

I. Introdução

As placas de trânsito são encontradas nas vias urbanas e estradas, e proveem regras de navegação que devem ser seguidas rigorosamente para evitar acidentes. O processo de detecção e reconhecimento de placas de trânsito pode ser utilizado em diversas aplicações, por exemplo: SADA (Sistemas de Assistência de Direção Avançada, que é de muito interesse da indústria e academia) [1], carros autônomos, mapeamento geográfico das placas trânsito, entre outras aplicações. Para aumentar a segurança dos motoristas, o sistema deve compreender o que está acontecendo ao seu redor [1]. Para essas aplicações, faz-se necessário a execução de um algoritmo em tempo real utilizando GPU (*Graphics Processing Unit*), TPU (*Tensor Processing Unit*) ou VPU (*Visual Processing Unit*), e em alguns casos podendo ser CPU (*Central Processing Unit*) dependendo do nível de otimização.

As placas de trânsito possuem características específicas que possibilitam sua identificação, sendo elas: formato, cores e símbolos. A partir dessas características é possível separar as placas de sinalização de trânsito em três espécies: de regulamentação, de advertência e de indicação.

D. C. Santos, Universidade do Oeste Paulista (Unoeste), Presidente Prudente, São Paulo, Brasil (dcastriani@gmail.com).

F. A. Silva, Universidade do Oeste Paulista (Unoeste), Presidente Prudente, São Paulo, Brasil (chico@unoeste.br).

D. R. Pereira, Universidade do Oeste Paulista (Unoeste), Presidente Prudente, São Paulo, Brasil (danilopereira@unoeste.br).

L. L. Almeida, Universidade do Oeste Paulista (Unoeste), Presidente Prudente, São Paulo, Brasil (llalmeida@uoneste.br).

A. O. Artero, Universidade Estadual Paulista (Unesp), Presidente Prudente, São Paulo, Brasil (almir.artero@unesp.br).

M. A. Piteri, Universidade Estadual Paulista (Unesp), Presidente Prudente, São Paulo, Brasil (marco.piteri@unesp.br)

V. H. C. de Albuquerque, Universidade de Fortaleza (Unifor), Fortaleza, Ceará, Brasil (victor.albuquerque@unifor.br).

Redes neurais convolucionais foram utilizadas em [1], [2], [4], [6], [12], [13], [17] para solucionar problemas de classificação de placas de trânsito, e utilizadas em [3], [7], [8], [9], [11], [16] para a classificação de outros tipos de objetos em imagens.

A detecção de objetos utilizando redes neurais foram utilizadas em [5], [6], [14]. Em [14] foi utilizada a detecção de objetos utilizando R-CNN. Em [5] foi utilizada uma Fast R-CNN que realiza a detecção de objetos em um tempo bem reduzido, e, de acordo com [6], quase atinge tempo real. Em [6], foi utilizada uma Faster R-CNN, que é mais rápida que as suas antecessoras.

Existem vários fatores que podem dificultar a detecção e o reconhecimento de placas de trânsito [1], [2], sendo eles: variação da iluminação, borramento na imagem, oclusão de galhos e folhas, diferentes pontos de vista, etc. De acordo com [12], a cor das placas de trânsito sofre alteração dependendo do clima, iluminação e período do dia (manhã, tarde e noite).

Em [1] e [2] foi utilizado o *dataset* GTSRB (*German Traffic Sign Recognition Benchmark*). Em [4] foi criado um *dataset* de *benchmark* 111 vezes maior do que o GTSDB (*German Traffic Sign Detection Benchmark*), e com uma resolução 32 vezes maior. Para a criação do *dataset* em [4], os autores propuseram adicionar várias condições adversas, como por exemplo, diferentes condições de iluminação e de clima. Também foram adicionadas imagens com oclusão.

Em [11] foi utilizado o *dataset* ILSVRC-2012, que possui 1.000 classes. Para treinamento do dataset foram utilizadas 1.300.000 imagens, para teste 100.000 imagens, e para validação 50.000 imagens.

Em [4] foram treinadas duas redes neurais diferentes, uma para detectar placas de trânsito, e outra para detectar e classificar. As redes utilizam a mesma estrutura, com exceção da última camada. Para aumentar a quantidade de dados, os autores usaram um *template* padrão para cada classe de sinal, rotacionando a imagem aleatoriamente entre -20º e 20º, também foi aplicada mudança de escala de forma aleatória usando um tamanho na faixa de 20 e 200, e distorção de perspectiva. Também foram adicionadas imagens sem placas de trânsito, e nessas imagens, foram adicionados ruídos.

Em [15] foi utilizado um aumento de dados, em que foram geradas imagens em tempo de execução do treinamento. Enquanto a rede estava sendo treinada utilizando a GPU, a CPU do computador realizava as transformações nas imagens utilizadas.

A metodologia desenvolvida neste trabalho é dividida em

duas etapas, sendo elas, a etapa de detecção da placa utilizando a Faster R-CNN (*Region-Based Convolutional Neural Network*) Inception-v2, e a etapa de reconhecimento utilizando uma CNN (*Convolutional Neural Network*). A etapa de detecção é responsável por indicar uma região de interesse onde pode existir uma placa de trânsito. Já a etapa de reconhecimento é responsável por indicar se aquela região de interesse, detectada na etapa anterior, possui ou não uma placa de trânsito, e se possuir, a rede CNN utilizada para o reconhecimento procura identificar qual placa está naquela região.

É sabido que o reconhecimento de placas de trânsito é uma tarefa muito importante para diversas aplicações, entretanto, em muitos casos faz-se necessária a implementação de algoritmos que apresentam tempos de processamento bem reduzidos com altas taxas de acerto, o que é um grande desafio. A motivação desse trabalho foi de utilizar redes neurais na busca de realizar a detecção e reconhecimento das placas de trânsito em tempo real com o maior número de acerto possível.

II. Referencial Teórico

Nessa seção são discutidos os fundamentos de CNN e Faster R-CNN que foram utilizados neste trabalho.

A. Convolutional Neural Network

CNN é uma rede neural que alterna entre camadas de convolução e de *pooling* [1], [3]. Após essas camadas, se encontra uma MLP (*Multi-Layear Perceptron*). O objetivo de uma rede neural convolucional é de utilizar uma imagem de tamanho predefinido e gerar uma saída indicando a classe desse objeto.

Um modelo de CNN em hierarquia foi proposto em [1], as placas foram separadas em categorias, e para cada categoria foi utilizada uma CNN diferente. O autor denomina a primeira CNN como Family Classifier CNN, que é responsável por classificar a categoria das placas, enquanto outras cinco CNNs denominadas Member Classifier CNN são responsáveis por classificar a placa de trânsito como mostrado na Fig. 1. A categoria da placa é definida de acordo com as suas características, podendo ser números, setas, formatos, entre outras.

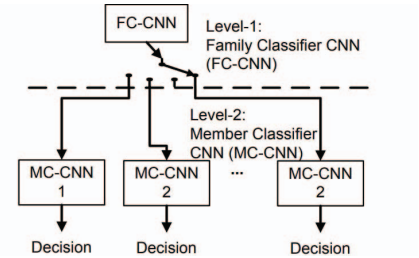


Fig. 1. Arquitetura em hierarquia utilizada em [1].

As CNNs normalmente utilizam várias camadas, sendo que uma camada comumente alimenta a próxima, de forma sequencial. Em [2] e [10] foram utilizadas arquiteturas não sequenciais. Em [2], no segundo estágio, foram utilizados dois fluxos em paralelo, sendo que apenas no primeiro fluxo existe uma camada de *pooling*, como mostrado na Fig. 2.

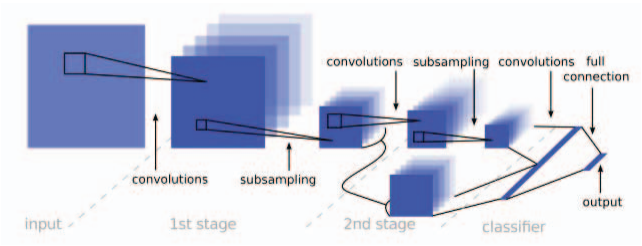


Fig. 2. Arquitetura utilizada em [2].

B. Camada de Convolução

Essa camada é responsável por gerar *feature maps* (mapas de características) [7]. Esses mapas são extraídos das imagens a partir de camadas de convolução.

Para cada unidade da camada, existe um filtro, que é uma matriz de tamanho predefinido [8]. Esse filtro é inicializado com valores aleatórios, que são ajustados no processo de treinamento [8].

Para gerar o *feature maps* é realizada uma operação de convolução entre a entrada e o filtro. O *stride* (passo) é um parâmetro que indica o quanto esse filtro irá se mover [7], [8]. Na Fig. 3, o *stride* é definido como 1x1. Ao se realizar uma operação de convolução, ocorre uma perda das bordas, e para contornar esse problema, é necessário realizar a operação de *zero-padding* [7], [8], que adiciona bordas com o valor 0, o tamanho das bordas depende do tamanho do filtro. A Fig. 4. representa uma operação de convolução utilizando *zero-padding* com um filtro de tamanho 3x3 e um *stride* de 1x1.

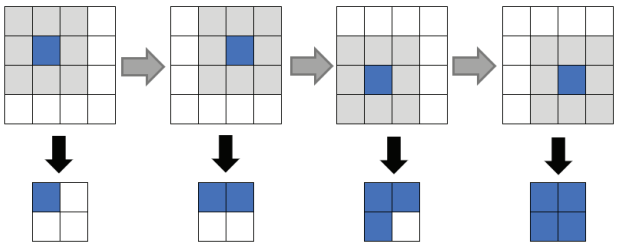


Fig. 3. Operação de convolução utilizando um *stride* de 1x1. As setas cinzas representam os passos, e as setas pretas representam a saída.

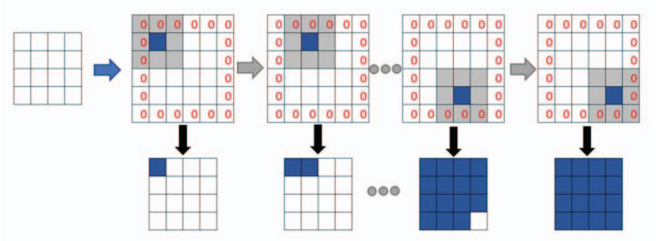


Fig. 4. Operação de convolução utilizando um *stride* de 1x1 após a operação de *zero-padding* representada pela seta azul. As setas cinzas representam os passos, e as setas pretas representam a saída.

Ao ser realizada a operação de convolução, os novos valores gerados passam por uma função de ativação antes de ir para a próxima camada. De acordo com [7] e [8], a função de ativação mais comum é a *Rectified Linear Unit* (ReLU), que foi utilizada em vários trabalhos [1], [7], [8], [10], [11], [18].

A definição da função de ativação ReLU é dada por (1), e é representada pela Fig. 5 [9].

$$ReLU(x) = \max(0, x) \quad (1)$$

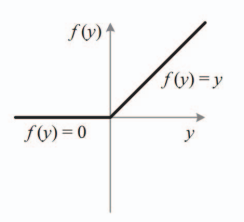


Fig. 5. Função de ativação ReLU [9].

C. Pooling

O objetivo da camada de *pooling* é utilizada para reduzir o tamanho original da sua entrada [7], [8]. Ao realizar essa redução, reduz-se o custo computacional da rede, e também evita o *over-fitting* [8].

Existem vários tipos de camadas de *pooling*. A camada *max-pooling* é a mais utilizada [7], [8], outro exemplo, é a camada *average pooling*.

Para a operação de *pooling*, é definida uma janela e o passo. Para cada janela, é definido um novo valor de acordo com o tipo de *pooling*. Esse novo valor fará parte da saída. Na Fig. 6, a matriz de entrada possui o tamanho 4x4, sendo definida uma janela com o tamanho 2x2 e o passo com o caminho 2x2. O resultado é uma saída com o tamanho 2x2, ou seja, a entrada foi reduzida em 75 %. O tamanho da altura e da largura podem ser calculados por (2) e (3) [8], onde *HP* é a nova altura, *WP* a nova largura, *H* a altura de entrada, *W* largura de entrada, *F* é o tamanho da janela e *S* é o passo.

$$HP = \frac{H-F}{S} + 1 \quad (2)$$

$$WP = \frac{W-F}{S} + 1 \quad (3)$$

A Fig. 6 mostra uma representação da operação de *max-pooling*. Essa operação consiste em selecionar o maior valor por janela [8]. Para se realizar a operação de *average pooling*, é calculada a média de cada janela, e os valores obtidos farão parte da saída. A operação de *pooling* da média também é um dos mais comuns [8], e é utilizado na arquitetura Inception-V4 [10].

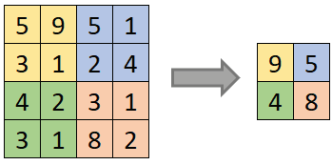


Fig. 6. A primeira matriz (a) representa a entrada, e a segunda matriz (b), a saída após uma operação de *max-pooling*.

D. Multi-Layer Perceptron

Após as camadas de convolução e de *pooling*, que realizam a extração de características, são utilizadas camadas MLP densas [8].

A MLP possui o objetivo de utilizar como entrada as características extraídas, e fornecer como saída a classe referente a imagem de entrada da rede. Cada camada da MLP possui um número *N* de unidades denominadas de neurônio. Esses neurônios são totalmente conectados com os neurônios da próxima camada, como mostrado na Fig. 7. Cada neurônio pode ser representado por (4), em que, x_j representa a entrada, w_{kj} representa o peso referente a ligação da entrada, e u_k representa o valor do neurônio [8].

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \quad (4)$$

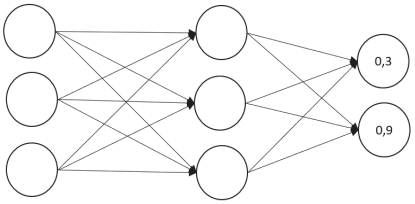


Fig. 7. Exemplo de MLP para classificar 2 objetos.

Por fim, é realizada a classificação na última camada utilizando a função de ativação *softmax* [8], que retorna a probabilidade utilizando os valores do neurônio de saída. A função *softmax* é representada por (5) [18], em que *n* é a quantidade de classes e x é o valor do neurônio.

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_{k=1}^{n} \exp(x_k)} \quad (5)$$

E. Fast R-CNN

A Fast R-CNN utiliza como entrada uma imagem inteira e utiliza uma série de regiões que supostamente podem conter objetos. Inicialmente, a rede processa a imagem com uma série de camadas de convolução e de *max pooling*, com objetivo de gerar *feature maps* (mapas de características). Para cada região proposta é extraído um vetor de tamanho fixo a partir do *feature map*. Cada vetor é utilizado para alimentar uma série de camadas totalmente conectadas. Por fim, são geradas duas saídas utilizando uma camada *softmax* para definir a classe de objeto, e um *boundbox regressor*, que retorna quatro valores contendo as coordenadas do objeto [5]. Na Fig. 8 é ilustrado um exemplo de arquitetura de uma Fast R-CNN.

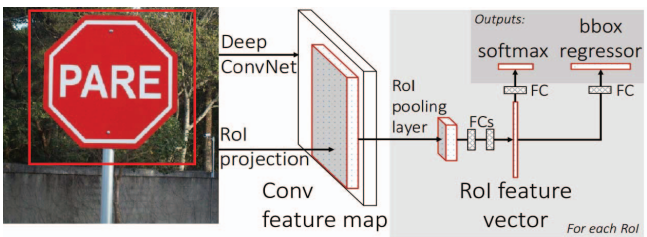


Fig. 8. De acordo com [5] o processo utiliza como entrada uma imagem, que é submetida a uma série de camadas de convolução e *pooling*, gerando *feature maps* (mapas de características). São extraídas regiões propostas, e para cada vetor de *feature map*, gera uma saída contendo a classe, e as coordenadas do objeto.

F. Faster R-CNN

A Faster R-CNN representada na Fig. 9 é uma evolução das redes R-CNN e Fast R-CNN. Essa rede é composta por dois módulos. O primeiro módulo é uma CNN profunda, sua função é propor regiões utilizando uma RPN (*Region Proposal Network*) [6]. O segundo módulo é um detector Fast R-CNN que utiliza as regiões propostas pela RPN.

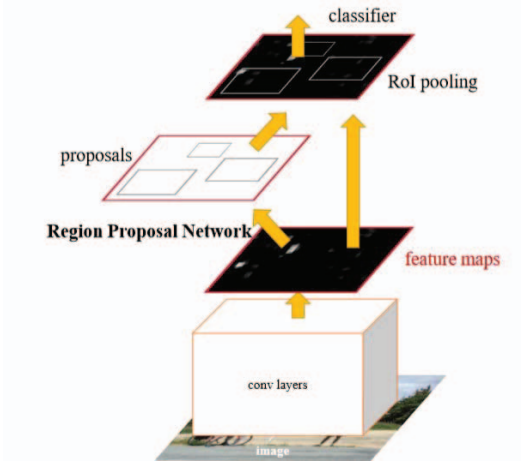


Fig. 9. De acordo com [6], a Faster R-CNN é uma rede unificada, e é utilizada para detectar objetos.

A RPN é uma rede totalmente convolucional que é treinada para gerar propostas de regiões de alta qualidade, que são usadas pela Fast R-CNN para a fase de detecção. A RPN compartilha a computação com a Fast R-CNN a partir de um conjunto comum de camadas de convolução. A RPN utiliza uma imagem como entrada, e fornece uma saída com um conjunto de regiões que podem conter objetos, cada um com sua pontuação.

Para gerar as propostas de região, uma pequena janela espacial $n \times n$ é deslizada sobre a saída do mapa de características convolucional (*conv feature map*) (Fig. 10). Em cada localização da janela deslizante, são previstas k propostas de regiões, de modo que a camada *reg* (*reg layer*) tem $4k$ saídas codificando as coordenadas das k caixas. A camada *cls* (*cls layer*) produz $2k$ pontuações de probabilidade de ter encontrado o objeto para cada proposta. As k propostas são parametrizadas em relação às k caixas de referência, chamadas de âncoras. As âncoras são os pontos centrais das janelas deslizantes, que são referências para ajudar a detectar objetos, com proporções e escalas de tamanhos diferentes.

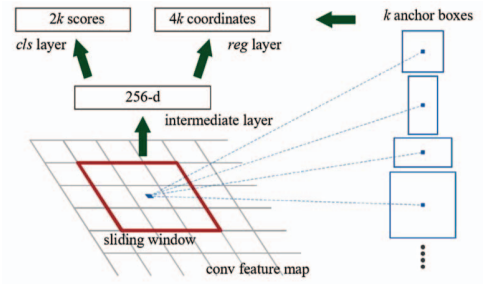


Fig. 10 Representação de âncoras na janela deslizante da *Region Proposal Network* (RPN) [6].

Após o processamento são retornadas a classe e a probabilidade de certeza de ter encontrado o objeto buscado, e as coordenadas de localização em uma região retangular.

III. Detecção e Reconhecimento de Placas de Trânsito

Nesta seção são discutidas as técnicas utilizadas na implementação da detecção e o reconhecimento das placas de trânsito.

A. Detecção

Na implementação realizada neste trabalho foi utilizada a biblioteca TensorFlow Object Detection API e uma rede Faster R-CNN com arquitetura Inception-v2.

Para a detecção, as placas de trânsito foram divididas em três classes, sendo elas: regulamentação, advertência e Pare. O critério utilizado para a determinação das classes foi correspondente a sua categoria, pois, cada categoria possui um formato, e conjunto de cores distintos. Algumas placas estão classificadas na Tabela I. O algoritmo é capaz de reconhecer 16 tipos de placas de regulamentação, 6 placas de advertência, e a placa de Pare. As placas utilizadas neste trabalho estão definidas na Fig. 11.

O detector recebe como entrada uma imagem inteira, essa imagem é processada pela rede, e por fim, é retornado um conjunto de dados contendo as regiões de interesse (RoIs – *Regions of Interest*). Para cada região de interesse que pode possuir um objeto, estão vinculadas seis informações, sendo elas: as coordenadas do canto superior esquerdo (x_0 e y_0), do canto inferior direito (x_1 e y_1), a classe correspondente ao objeto de acordo com a Tabela I, e a porcentagem indicando a probabilidade de a classe estar correta.

TABELA I
CLASSES DA REDE DE DETECÇÃO

Classe	Amostras de Placas
regulamentação	
advertência	
pare	

Para decidir qual região de interesse será utilizada ou não, é definido um parâmetro *threshold*. Se a porcentagem for menor que o *threshold*, a região de interesse é descartada, caso contrário, essa região de interesse será utilizada na próxima etapa.

B. Reconhecimento

Para realizar a classificação, foi implementada uma CNN baseada na arquitetura VGG16 configuração D [11]. Apenas foi reduzida a quantidade de parâmetros de algumas camadas. Também foi implementada a arquitetura Inception-v4 [10] reduzindo a quantidade de módulos. O objetivo de reduzir os parâmetros das camadas da arquitetura VGG 16 e a quantidade de módulos da Inception-v4, foi para diminuir o uso de memória e o tempo de treinamento.

Na Tabela II é possível visualizar as alterações realizadas. Para o reconhecimento, existem 24 classes, sendo 23 placas de trânsito de acordo com o significado, e uma classe especial que indica quando não há placas de trânsito, como mostrado na Fig. 11.

TABELA II
VGG16 X ARQUITETURA UTILIZADA

VGG16 Configuração D	Arquitetura Utilizada
Conv3 – 64	Conv3 – 64
Conv3 – 64	Conv3 – 64
max-pooling	
Conv3 – 128	Conv3 – 128
Conv3 – 128	Conv3 – 128
max-pooling	
Conv3 – 258	Conv3 – 258
Conv3 – 258	Conv3 – 258
Conv3 – 258	
max-pooling	
Conv3 – 512	Conv3 – 256
Conv3 – 512	Conv3 – 256
Conv3 – 512	Conv3 – 256
max-pooling	
Conv3 – 512	Conv3 – 256
Conv3 – 512	Conv3 – 256
Conv3 – 512	Conv3 – 256
max-pooling	
FC - 4096	FC – 2048
FC - 4096	FC – 2048
FC - 1000	FC – 2048
	FC – 24
Softmax	

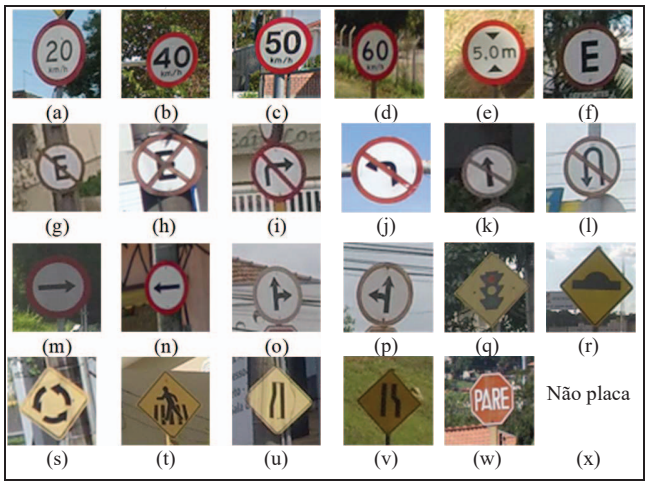


Fig. 11. Classes de placas de trânsito utilizadas no reconhecimento: (a) 20 km/h; (b) 40 km/h; (c) 50 km/h; (d) 60 km/h; (e) altura 5,0 m; (f) estacionamento permitido; (g) proibido estacionar; (h) proibido parar e estacionar; (i) proibido virar à direita; (j) proibido virar à esquerda; (k) sentido proibido; (l) proibido retornar; (m) sentido direita; (n) sentido esquerda; (o) siga em frente ou direita; (p) siga em frente ou esquerda; (q) semáforo; (r) lombada; (s) rotatória; (t) faixa de pedestre; (u) estreitamento à esquerda; (v) estreitamento à direita; (w) pare; (x) não placa.

A rede possui uma entrada fixa de 128x128x3, sendo necessário realizar um redimensionamento nas imagens de entrada.

Ao realizar o reconhecimento, a rede retorna um conjunto de dados contendo a probabilidade de cada classe especificada na Fig. 11. A classe com a maior probabilidade é escolhida, e a imagem é classificada.

C. *Treinamento do Detector*

Foi criado um *dataset* [19] para o treinamento da rede Faster R-CNN contendo 5.268 imagens (originais e rotacionadas). Nesse *dataset* foi utilizada a técnica de aumento de dados, que consiste em aplicar filtros e transformações nas imagens. Foram aplicadas rotações de 2º, 3º, 5º e 7º no sentido horário e também no sentido anti-horário. A partir do aumento de dados, 90% das imagens foram destinadas ao treino, e 10 % das imagens destinadas à validação da rede.

D. *Treinamento da CNN para Reconhecimento*

Para construir o *dataset* [19] de treinamento e validação das redes CNN para reconhecimento, foram segmentadas as placas de trânsito das imagens originais e rotacionadas (6.731 placas segmentadas). Foi criado um *script* que realiza a segmentação de todas as placas de cada imagem, com base nas extrações realizadas na construção do *dataset* de detecção. As placas obtidas foram separadas em pastas devidamente rotuladas com os nomes das classes.

Após esse processo, foi realizado o aumento de dados, e para isso, as imagens de algumas classes foram espelhadas horizontalmente, por exemplo, a placa de lombada (Fig. 11 (r)) e semáforo (Fig. 11 (q)). Isto foi feito para aumentar a quantidade de imagens com essas placas, já que esse espelhamento não altera o significado do sinal.

O espelhamento também foi aplicado nas placas com indicação sentido de circulação da via (Fig. 11 (m, n)), de seguir à frente ou à esquerda (Fig. 11 (p)) e também de seguir à frente ou à direita (Fig. 11 (o)), gerando novas imagens com significados opostos.

Algumas imagens não podem ser espelhadas, por exemplo, proibido virar à esquerda (Fig. 11 (j)), e proibido virar à direita (Fig. 11 (i)), pois, possuem uma faixa na diagonal que vai de cima para baixo da esquerda para direita. Se realizar o espelhamento nas imagens dessas placas, o sinal ficaria errado.

As imagens foram divididas em dois grupos distintos, segundo os mesmos critérios para a criação dos *datasets* de treinamento e validação utilizados na detecção.

E. *Obtenção das Imagens*

Para realizar a obtenção das imagens foi utilizada uma câmera GoPro Hero 5 Black. A câmera foi fixada no para-brisa de um carro, e foram gravados vídeos nas principais ruas e avenidas da cidade de Presidente Prudente. Os vídeos foram gravados na resolução 1920 x 1080 pixels. Para cada vídeo gravado, foram extraídos *frames* contendo as placas escolhidas para compor os *datasets* de detecção e de reconhecimento.

F. *Detecção e Reconhecimento*

O fluxo de execução do algoritmo para detecção e reconhecimento de placas de trânsito (Fig. 12) utilizado nos experimentos funciona da seguinte forma: primeiramente, é obtido um *frame* a partir de um dos vídeos gravados. Esse vídeo não foi utilizado na extração de nenhuma das imagens utilizadas para construir os *datasets* para treino das redes. Cada *frame* é utilizado como entrada do detector de placas, que retorna um conjunto de regiões de interesse. Para cada região de interesse são obtidas as coordenadas da placa, a classe, e uma probabilidade.

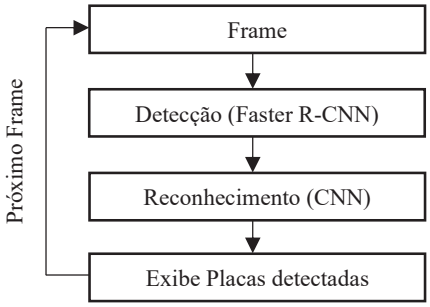


Fig. 12. Fluxo de execução do algoritmo.

Cada região de interesse contendo placa com probabilidade maior que 30% (valor de *threshold* definido empiricamente) foi adicionada em uma lista de possíveis placas. Todas as regiões de interesse da lista servem de entrada para o algoritmo que implementa a CNN de reconhecimento. O algoritmo retorna a classe de cada placa, conforme indicado na Fig. 11, e uma pontuação que indica a probabilidade de ser a classe da placa reconhecida (Fig. 13).

Se a classe retornada for "não placa", a região de interesse não será exibida, caso contrário, será exibida a região com a cor correspondente às classes da detecção (verde para advertência, amarelo para regulamentação, e vermelho para Pare), juntamente com o título da classe retornada pelo algoritmo de reconhecimento, e a sua respectiva porcentagem. Na Fig. 13 (a) é mostrada uma imagem com quatro placas detectadas e reconhecidas, em (b), (c), (d) e (e) estão as regiões das placas ampliadas.

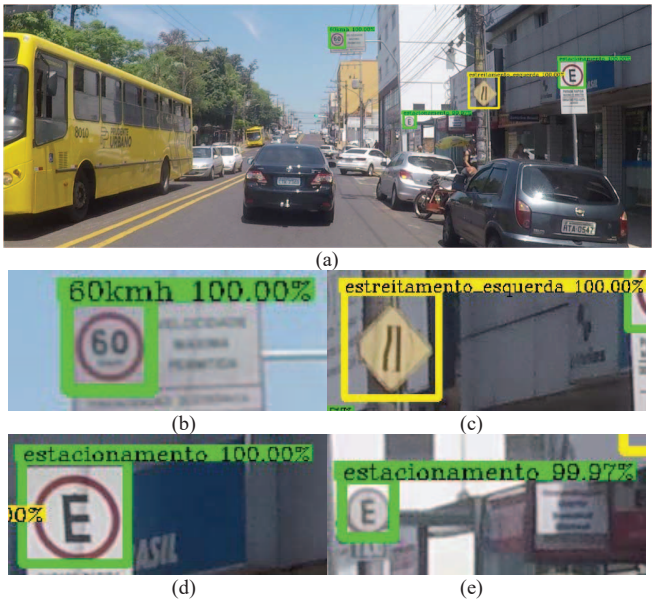


Fig. 13. Detecção e reconhecimento utilizando um *frame* como entrada. (a) representa a imagem completa, (b), (c), (d) e (e) mostram as regiões ampliadas das placas detectadas e reconhecidas.

G. Falhas

Na Fig. 14 são apresentadas algumas falhas que ocorreram na detecção das placas, na detecção e reconhecimento e apenas no reconhecimento. Na Fig. 14 (a) tem-se uma falha na etapa de detecção pela rede Faster R-CNN, em que a placa indicando proibido parar e estacionar não foi detectada. Na Fig. 14 (b) houve uma falha na detecção de uma falsa placa de trânsito e posteriormente uma falha no reconhecimento, que ao invés da rede CNN ter indicado como sendo não placa, acabou reconhecendo como placa de Pare. Algo semelhante ocorreu com a Fig. 14 (c), porém, a rede reconheceu a falsa placa como sendo uma placa de proibido retornar. Na Fig. 14 (d) ocorreu uma falha no reconhecimento, em que a rede não identificou corretamente a placa de velocidade. Uma outra falha no reconhecimento ocorreu na Fig. 14 (e), porém com a identificação de uma placa bem diferente da que ela representa.

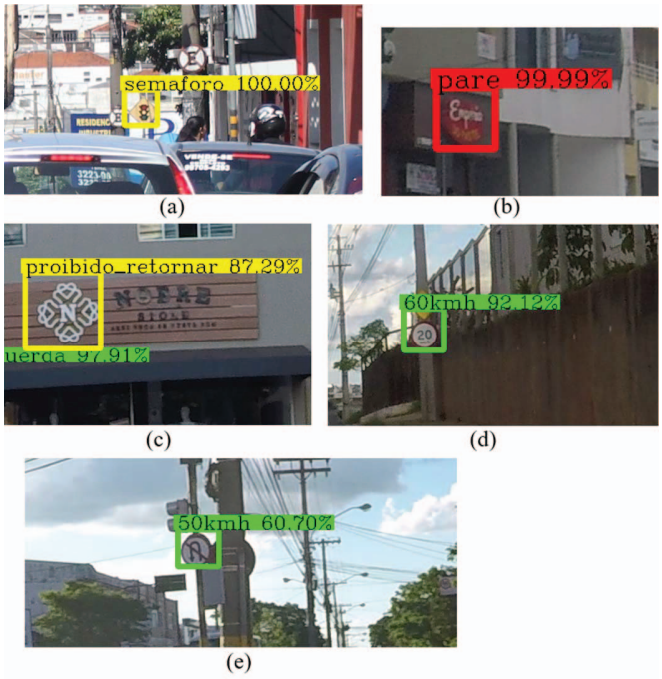


Fig. 14. Algumas falhas que ocorreram na detecção das placas, na detecção e reconhecimento e apenas no reconhecimento.

IV. Experimentos Realizados

O *hardware* utilizado tanto para treino quanto para testes foi um computador com processador Intel I5-4460, 8GB de RAM, placa de vídeo Nvidia Geforce GTX 1060 com 6GB, e SSD. Também foram realizados testes com um notebook Dell Inspiron 15 7559 com processador Intel I5-6300 HQ, 10GB de RAM, placa de vídeo GTX 960M com 4GB, e SSD. Foi utilizado o sistema operacional Ubuntu 18.04 LTS em ambos os computadores. Todos os testes foram realizados com imagens de resolução de 1920 x 1080 pixels.

A tabela III mostra os resultados da detecção para 107 imagens contendo 159 placas, que foram escolhidas aleatoriamente. Vale ressaltar que as imagens escolhidas não estão em nenhum *dataset* utilizado para treino. Essas imagens utilizadas nos experimentos e também os resultados podem ser encontrados em [19].

Na Tabela III a coluna "Acertos" indica que a placa foi detectada corretamente, A coluna "Erros" indica que foram detectadas regiões que não possui placa, e a coluna "Não detectadas" indica que existia placa na imagem, mas a rede não foi capaz de detectá-la.

TABELA III
RESULTADOS DA DETECÇÃO NAS 107 IMAGENS

	Acertos	Erros	Não detectadas
Placas	148	11	12
Porcentagem	87,83%	5,82%	6,34%

A tabela IV contém os resultados do reconhecimento das placas a partir das regiões válidas (regiões que realmente contêm placas) detectadas na etapa anterior (etapa de detecção).

TABELA IV
RECONHECIMENTO POR CLASSE

	Arquiteturas	VGG 16 Custom		Inception v4 Custom	
Classes	Placas nas imagens	Acertos	Erros	Acertos	Erros
(a)	1	0	1	0	1
(b)	3	0	3	0	3
(c)	0	0	0	0	0
(d)	4	2	2	2	2
(e)	0	0	0	0	0
(f)	20	13	7	18	2
(g)	8	6	2	7	1
(h)	7	4	3	5	2
(i)	11	6	5	8	3
(j)	8	6	2	7	1
(k)	1	0	1	0	1
(l)	19	17	2	18	1
(m)	1	0	1	1	0
(n)	3	2	1	3	0
(o)	6	4	2	6	0
(p)	9	9	0	9	0
(q)	3	3	0	3	0
(r)	6	5	1	6	0
(s)	4	4	0	4	0
(t)	4	3	1	3	1
(u)	0	0	0	0	0
(v)	1	1	0	1	0
(w)	27	26	1	26	1
(x)	13	6	7	5	8
Total	159	117	42	132	27

É possível realizar o comparativo entre as duas arquiteturas utilizadas, em que foi possível verificar que a arquitetura Inception-v4 obteve melhores resultados que a arquitetura VGG 16. Na Tabela V é possível ter uma visão geral em relação ao reconhecimento das placas, comparando-se as arquiteturas.

TABELA V
COMPARATIVO DAS ARQUITETURAS DE CNN UTILIZADAS NO RECONHECIMENTO

	Acertos	Erros	Acertos %
VGG 16 Custom	117	42	73,58%
Inception-v4 Custom	132	27	83,02 %

A Tabela VI mostra um comparativo do tempo de execução do algoritmo de detecção (usando a rede Faster R-CNN) e reconhecimento (usando a rede CNN) de placas de trânsito. Foram realizados testes de tempo das redes de detecção e reconhecimento separadamente, e em conjunto. Para realizar o cálculo dos tempos (em milissegundos) foram calculados os tempos médios das 10 primeiras imagens.

TABELA VI
COMPARATIVO DE TEMPO DE EXECUÇÃO EM MILISSEGUNDOS

	Detecção	Reconhecimento	Conjunto
Computador	86,6	5,3	226,2
Notebook	230,6	10,8	408,9

V. CONCLUSÕES

A partir dos experimentos realizados, observou-se que para treinar as CNNs adotadas, quanto maior o número de imagens de uma mesma placa de trânsito com características semelhantes e fundo diferente, maior é a acurácia da rede. Também se tornou necessário adicionar uma classe especial na etapa de reconhecimento, a classe "não placa", pois, a região de interesse pode não conter placa, ou a placa não está presente no *dataset*. Com os resultados obtidos, é possível verificar que das arquiteturas utilizadas para se realizar o reconhecimento das placas de trânsito, a que apresentou melhores resultados foi a Inception-v4.

Como trabalhos futuros, seria de grande valia adicionar novos objetos para se realizar a detecção e o reconhecimento, como, por exemplo, sinais localizados no chão, outros tipos de sinais de trânsito, e aumentar o número imagens nos *datasets*.

REFERÊNCIAS

[1] X. Mao, S. Hijazi, R. Casas, P. Kaul, R. Kumar and C. Rowen, "Hierarchical CNN for traffic sign recognition," *2016 IEEE Intelligent Vehicles Symposium (IV)*, Gothenburg, 2016, pp. 130-135.

[2] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale Convolutional Networks," *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, 2011, pp. 2809-2813.

[3] P. Sermanet, S. Chintala and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, Tsukuba, Japan, 2012, pp. 3288-3291.

[4] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li and S. Hu, "Traffic-Sign Detection and Classification in the Wild," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Las Vegas, NV, 2016, pp. 2110-2118.

[5] R. Girshick, "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 1440-1448.

[6] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 1 June 2017.

[7] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, Antalya, 2017, pp. 1-6.

[8] F. H. D. Araújo, A. C. Carneiro, R. R. V. Silva, F. N. S. Medeiros and D. M. Ushizima, "Redes Neurais Convolucionais com Tensorflow: Teoria e Prática," in *2017 III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos*, v. 1, n. 1, p. 382-406

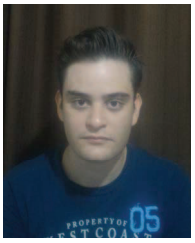
[9] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 1026-1034.

[10] C. Szegedy, S. Ioffe, V. Vanhoucke, A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *AAAI* Vol. 4. 2017 P.12

[11] K. Simonyan, A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition" 2015. [Online] Available: https://arxiv.org/pdf/1409.1556.pdf

[12] Y. Wu, Y. Liu, J. Li, H. Liu and X. Hu, "Traffic sign detection based on convolutional neural networks," *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, TX, 2013, pp. 1-7.

[13] Y. Yang, H. Luo, H. Xu and F. Wu, "Towards Real-Time Traffic Sign Detection and Classification," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 7, pp. 2022-2031, July 2016.
[14] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142-158, 1 Jan. 2016.
[15] A. Krizhevsky, I. Sutskever, Geoffrey E. Hinton "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems*, vol. 1, pp. 1097-1105, Dec 2012.
[16] A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in *ISVC*, pages 867-877, 2015
[17] S. Jung, U. Lee, J. Jung and D. H. Shim, "Real-time Traffic Sign Recognition system with deep convolutional neural network," *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Xi'an, 2016, pp. 31-34.
[18] M. Y. W. Teow, "Understanding convolutional neural networks using a minimal model for handwritten digit recognition," *2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, Kota Kinabalu, 2017, pp. 167-172.
[19] *Datasets* de imagens usadas no treinamento das redes para detecção e reconhecimento de placas de trânsito. [ONLINE] Available: https://drive.google.com/drive/folders/1HMa5IQSCCpT2oHlDhbIfXPzrixbUpvjJ?fbclid=IwAR3Vb2oRrJxGDpIczZIa27nlQX8jNuUbM5sAC5mwMHV-kdS71kMdO5WSAc8.



Daniel Castriani Santos graduado em Ciência da Computação na Universidade do Oeste Paulista (Unoeste), Brasil (2018).

Francisco Assis da Silva graduado em Ciência da Computação na Universidade do Oeste Paulista (Unoeste), Brasil (1998), Mestre em Ciência da Computação na Universidade Federal do Rio Grande do Sul (UFRGS), Brasil (2002), Doutor em Ciências, Programa de Engenharia Elétrica da Universidade de São Paulo (USP), Brasil (2012) e Pós-Doutor na Universidade Estadual Paulista (Unesp), Brasil (2017). Atualmente é professor na Universidade do Oeste Paulista (Unoeste), Brasil.

Danillo Roberto Pereira graduado em Ciência da Computação na Universidade Estadual de São Paulo (Unesp), SP, Brasil em 2006, Mestre em Ciência da Computação na Universidade de Campinas (Unicamp), Brasil (2009), Doutor em Ciência da Computação (Unicamp), Brasil (2013), Pós-Doutor na Universidade Estadual de São Paulo, Brasil (2016) e Pós-Doutor na Universidade Federal de São Carlos (UFSCar), Brasil (2017). Atualmente é professor na Universidade do Oeste Paulista (Unoeste), Brasil.

Leandro Luiz de Almeida graduado em Ciência da Computação na Universidade do Oeste Paulista (Unoeste), Brasil (1997), Mestre em Ciências Cartográficas na Universidade Estadual de São Paulo (Unesp), Brasil (2001) e Doutor em Ciências, Programa de Engenharia Elétrica da Universidade de São Paulo (USP), Brasil (2013). Atualmente é professor na Universidade do Oeste Paulista (Unoeste), Brasil.

Almir Olivette Artero graduado em Matemática na Universidade Estadual de São Paulo (Unesp), Brasil (1990), Mestre em Ciências Cartográficas na Universidade Estadual de São Paulo (Unesp), Brasil (1999) e Doutor em Ciência da Computação na Universidade de São Paulo (USP) (2005). Atualmente é professor na Universidade do Estado de São Paulo (Unesp), Brasil.

Marco Antônio Piteri graduado em Matemática na Universidade Estadual de São Paulo (Unesp), Brasil (1983), Mestre em Engenharia de Sistemas e Computação na Universidade Federal do Rio de Janeiro (UFRJ), Brasil (1988) e Doutor em Engenharia Civil na Universidade Técnica de Lisboa (UTL), Portugal (1996). Atualmente é professor na Universidade do Estado de São Paulo (Unesp), Brasil.

Victor Hugo C. de Albuquerque graduado em Tecnologia Mecatrônica no Centro Tecnológico Federal de Educação do Ceará, Brasil (2006), Mestre em Engenharia Teleinformática na Universidade Federal do Ceará, Brasil (2007) e Doutor em Engenharia Mecânica com ênfase em Materiais na Universidade Federal da Paraíba, Brasil (2010). Atualmente é professor do Programa de Graduação em Informática da Universidade de Fortaleza (Unifor), Brasil.