Desenvolvimento de um Sistema de Detecção de Placas Licenciadas

Rafael A. A. Tomé[1], **Arnold C. V. Lima**[1], **Gustavo T. Laureano**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 - CEP 74001-970 – Goiânia – GO – Brazil

{noprafa, arnoldcvl, gustavoengdm}@gmail.com

Abstract. *The automatic detection of license plates through images has been the subject of study for many years, but to this date there are no robust enough methods to handle different poses or that handle complex scenes in a satisfactory way. One of the main difficulties about the construction of a license plate detection system is modeling the visual pattern to be detected, this pattern can be ambiguous and have variable point of view, besides varying according to the legislation. This paper presents a method of detecting vehicle license plates using Haar features and Adaboost algorithm, which does not require direct modeling or definition of characteristics for the classification process. The proposed methodology was evaluated using the database from UFPR (Federal University of Paraná) and the results show the viability of the proposal.*

Resumo. *A detecção automática de placas de veículos por imagens tem sido objeto de estudo há muitos anos, mas até hoje não existem métodos robustos o suficiente para lidar com diferentes poses ou que lidam com cenas complexas de modo satisfatório. Uma das principais dificuldades na construção de um sistema de detecção de placas é a modelagem do padrão visual que se deseja detectar, sendo este padrão ambíguo e variável com o ponto de vista, além de variar de acordo com a legislação. Este trabalho apresenta um método de detecção de placas de veículos usando características de Haar e Adaboost, sendo desnecessária a modelagem direta ou a definição de características para o processo de classificação. A metodologia proposta foi avaliada usando a base de dados oriunda da UFPR (Universidade Federal do Paraná) e resultados mostram a viabilidade da proposta.*

1. Introdução

A placa de licenciamento veicular é o mecanismo atual de identificação de veículos. Elas são normalmente formadas por uma sequência de letras e dígitos, dispostas de modo horizontal e delimitadas por uma borda retangular. A sequência de letras e números é única, mas o padrão visual pode variar de acordo com a legislação de cada região.

A detecção e reconhecimento de placas de veículos a partir de imagens tem sido colocado como um assunto de grande interesse em áreas como a de transporte e segurança, principalmente pelo intenso aumento do fluxo de veículos e as próprias limitações humanas no registro e observação de imagens. No entanto, apesar de não ser um assunto recente, até hoje não existem métodos robustos o suficiente para lidar com diferentes poses

ou que lidam com cenas complexas[1] de modo satisfatório.

Os sistemas que se propõem a realizar a automatização do reconhecimento de placas são chamados sistemas ALPR (*Automatic License Plate Recognition*). Esses sistemas fazem uso de conceitos de Processamento de Imagens e Reconhecimento de Padrões para a detecção e reconhecimento das placas em uma imagem [[Szeliski 2011], [Wang et al. 2012]].

O problema da detecção e identificação de placas consiste em várias etapas [Du et al. 2013], como é mostrado na Figura 1, que podem ser generalizadas entre os seguintes passos: Pré-processamento da imagem: essa etapa consiste no tratamento de eventuais ruídos como sombreamento, iluminação, baixa qualidade de captura da câmera e afins; Extração de características: dadas as características do objeto, procura-se na imagem aquela com maior probabilidade de conter o objeto procurado e separar do restante da imagem; Segmentação de caracteres: uma vez que a região desejada foi extraída, pode-se iniciar a procura por eventuais caracteres que compõem a placa veicular, ou seja, caracteres alfabéticos e numéricos, e então extrai-los para a próxima fase; Reconhecimento dos caracteres: partindo da fase anterior, pode-se começar a parte final, que consiste no reconhecimento dos caracteres extraídos e, assim, realizar o registro do veículo em um dado banco de dados para as possíveis aplicações.

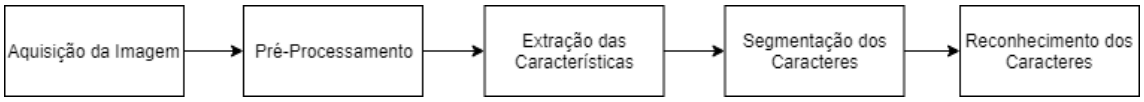

Figura 1. Pipeline de um sistema ALPR

Dentre as etapas citadas, a detecção da região de placa é uma das mais desafiadoras, dada a variabilidade visual que as placas podem assumir, além de serem formadas majoritariamente de características com grande propensão a ambiguidades em um ambiente antrópico.

Na literatura é possível encontrar diferentes meios e estratégias para se detectar placas em uma imagem. Em [Quiros et al. 2017], são utilizadas técnicas de processamento de imagens como a detecção de bordas e correspondência de contorno (*Contour Matching*), e utilização parâmetros como área e proporção para filtrar os contornos da imagem. Outra abordagem semelhante ocorre em [Ahn et al. 2017], onde o autor combina detecção de borda de Canny e bordas Laplacianas para tentar melhorar a taxa de acerto de métodos de detecção de placa que exploram as bordas. Em [AlyanNezhadi et al. 2017], o autor vai além e propõe a utilização de fortes ferramentas como o Filtro Gaussiano e a Rede Bayesiana na fase de pré-processamento da imagem, com a finalidade de melhorar a informação das bordas da placa em cénarios de iluminação ruim e fundos complexos, e por fim realiza a detecção de bordas.

Um sistema ALPR é capaz de identificar a região em que a placa se encontra e, após isto, identificar quais são os caracteres presentes nela. A finalidade é que esta extração e reconhecimento possa ser utilizada em outras aplicações, como monitoramento de entradas e saídas [Chandra et al. 2017] de uma propriedade qualquer, como a própria

[1]Neste trabalho uma *cena complexa* é entendida como uma imagem com grande quantidade de informações visuais variadas.

universidade.

O objetivo deste trabalho é o estudo e desenvolvimento de um sistema de detecção de regiões de placas em imagens, tendo como objetivo final a construção de um Sistema de Detecção Automática de Placas Licenciadas que é uma das principais fases de um sistema ALPR. Para isso, a metodologia proposta baseia-se no uso de características de Haar e no emprego do meta-algoritmo Adaboost para criar uma cascata de classificadores que melhor detectam a região de placa presente na imagem. Tal metodologia foi escolhida pois mesmo com o treinamento sendo uma etapa que exige muito tempo e recurso computacional, o reconhecimento é extremamente rápido podendo ser utilizado em máquinas com menor poder de processamento. O projeto foi desenvolvido com ênfase na fase de detecção da região em que a placa do veículo se encontra na imagem.

2. Materiais e Métodos

O padrão da placa brasileira pode ser visto na Figura 2. Este modelo foi definido em 1990 e está em vigor desde então, sendo constituído de 3 caracteres alfabéticos, seguidos de um hífen e mais 4 caracteres numéricos com o tamanho padrão de 400 milímetros de largura por 130 milímetros de altura.

Figura 2. Placa Brasileira

Atualmente, cada carro possui uma placa na parte frontal e uma na traseira, e esta informação está disponível por meio de imagens ou de vídeos. A detecção da região em que uma placa se encontra em uma determinada imagem configura um problema de reconhecimento de padrões, dado como entrada uma imagem digital e como saída a posição da placa em coordenadas de imagem.

Reconhecimento de padrões é a área da ciência da computação que busca estimar um modelo matemático/computacional que consiga relacionar padrões de entradas e suas respectivas saídas dado um conjunto de dados de treinamento [Gibson 2005].

2.1. Haar Cascade

O *Haar Cascade* é um método de detecção de padrões proposto por Paul Viola e Michael Jones [Viola and Jones 2001] utilizado para a detecção de possíveis regiões de placa na imagem. É um metodo eficaz de detecção de objetos e se tornou conhecido pelo seu bom desempenho no problema de detecção de faces humanas em imagens. É uma abordagem de *Machine Learning* em que, dado um conjunto de imagens de treinamento, composto de imagens positivas (que contém o padrão de interesse) e imagens negativas (que não possuem o padrão de interesse), o processo de treinamento encontra uma função formada por uma cascata de classificadores com base nas características de Haar. Um pipeline básico do processo de treinamento pode ser visto na Figura 3.

Inicialmente, para realizar o treinamento do classificador, o algoritmo necessita de uma quantidade considerável de imagens positivas, que possuem o objeto de interesse,

e de imagens negativas, que não possuem o objeto de interesse. Definido o conjunto de imagens para realizar o treinamento, o próximo passo é a extração de características. Para este fim, o algoritmo utiliza as *Haar-like features*, que são estruturas que representam diferenças entre as somas de regiões retangulares em uma janela de observação, assumindo tamanhos e formas variadas, assim como apresentado da Figura 4.

A partir de uma característica Haar é possível obter um simples valor númerico subtraindo a soma dos pixels sobre a área branca e a soma dos pixels sobre a área preta. Este valor é calculado para uma vasta coleção de características de Haar que podem ser criadas na mesma dimensão da imagem, assumindo tamanhos, formas e posições diferentes. Essa flexibilidade resulta em um crescimento exponencial da quantidade de características de Haar em função do tamanho da imagem. Uma imagem com 24 linhas e 24 colunas, por exemplo, pode resultar aproximadamente em 160 mil variações das características de Haar apresentadas na Figura 4.

Para o cálculo mais eficiente das características de Haar os autores de [Viola and Jones 2001] utilizam o conceito de Imagem Integral, que torna o cálculo das somas das áreas em um problema que pode ser resolvido em tempo constante, sendo independente do tamanho da característica.

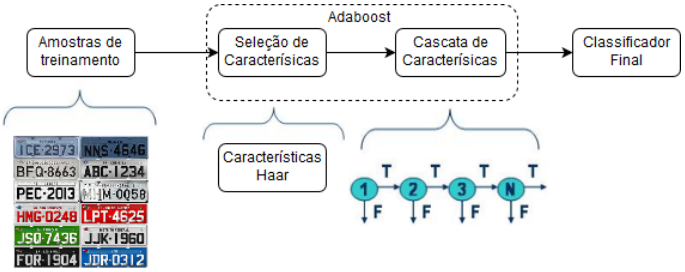

Figura 3. Pipeline do treinamento de uma função Haar

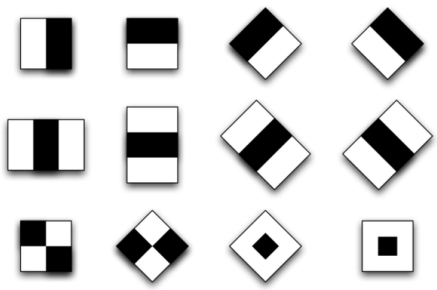

Figura 4. Exemplos de características Haar

2.2. Adaboost

Diante da diversidade de características de Haar possíveis, algumas podem se mostrar irrelevantes perante ao objeto de interesse durante a fase de treinamento, sendo possível então reduzir a quantidade de características. Para isso, emprega-se o algoritmo Adaboost [Freund and Schapire 1997].

O Adaboost é um meta-algoritmo que constrói um classificador forte a partir da combinação de vários classificadores fracos. Estes últimos recebem esse nome pois sozinhos não são capazes de classificar uma imagem. Os classificadores fracos são construídos a partir do processo de treinamento do algoritmo, onde são calculadas as respostas para as características Haar nas amostras de imagens utilizadas no treinamento e são mantidas as características com o menor número de erros no processo de classificação.

A cada iteração do treinamento, os pesos de cada exemplo classificado incorretamente é aumentado (ou, alternativamente, os pesos classificados corretamente são decrementados), para que então o novo classificador trabalhe em mais exemplos na etapa seguinte.

Como o AdaBoost é um algoritmo baseado em *Machine Learning*, o primeiro passo consiste na fase de treinamento, para que possamos posteriormente reconhecer objetos. A entrada inicial do algoritmo é um conjunto de amostras, no formato $S = (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_m, y_m)$

Onde cada x_i representa todas as respostas de Haar para uma determinada imagem, enquanto y_i representa a resposta atual do classificador, no caso se existe ou não placa. O total de amostras, no caso o total de imagens, é igual a m. Na primeira etapa do algoritmo, cada conjunto de dados do treinamento, recebe o mesmo peso, no caso:

$$D_0 = \frac{1}{m} \tag{1}$$

Em cada iteração do algoritmo, um classificador fraco- ou hipótese atual h_t- é gerado, baseado nos pesos de cada conjunto de atributos x_i, com foco na classificação correta dos dados com maior peso. Então, a cada iteração, o objetivo do classificador fraco é minimizar o erro do treinamento e_t, dado por:

$$e_t = \sum_{i:h_t(x_t)\neq y_t} D_t(i) \tag{2}$$

Então , os pesos são modificados para aumentar o peso dos dados incorretamente classificados e um novo estágio se inicia. Após todos os estágios terminarem, o AdaBoost combina cada um dos classificadores fracos h_t para formarem um classificador forte $H(x)$ no final, capaz de predizer se o objeto de interesse está ou não na imagem, onde uma votação ponderada é realizada com o parâmetro γ_i representa a importância de cada classificador fraco.

No processo final do treinamento, tem-se o que é chamado de ”cascata de classificadores”, onde existem vários classificadores fortes organizados de forma sequencial, formando uma cascata. Essa cascata é, então, o classificador final resultante. Pra termos o resultado de uma classificação, aplicamos então essa cascata de classificadores em uma janela, e, caso ela falhe em detectar o objeto de interesse em algum dos classificadores fracos da cascata, a janela é descartada. A janela é classificada como o objeto de interesse caso ela passe por todos estágios da cascata de classificadores, como se pode observar na Figura 6.

Mais detalhes sobre o conjunto de imagens usada no treinamento da função cascade utilizada neste trabalho serão fornecidas na próxima seção.

1: Dado um conjunto de treinamento $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, com $y_i \in \{-1, +1\}$
2: Inicializar $D_t(\mathbf{x}_i) \leftarrow \frac{1}{n}$ e $t \leftarrow 1$
3: **repita**
4: Treinar o algoritmo base utilizando a distribuição D_t
5: Calcular o erro ϵ_t da hipótese h_t gerada

$$\epsilon_t \leftarrow \Pr_{i \sim D_t}\left[h_t(\mathbf{x}_i) \neq y_i\right] = \sum_{i:h_t(\mathbf{x}_i) \neq y_i} D_t(\mathbf{x}_i)$$

6: Escolher $\gamma_t \leftarrow \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ e $Z_t \leftarrow \sum_{i=1}^{n} D_t(\mathbf{x}_i)\exp\left(-\gamma_t y_i h_t(\mathbf{x}_i)\right)$, em que Z_t é um termo normalizador
7: Atualizar D_t:

$$D_{t+1}(\mathbf{x}_i) \leftarrow \frac{D_t(\mathbf{x}_i)}{Z_t} \times \begin{cases} \exp(-\gamma_t) & se\, h_t(\mathbf{x}_i) = y_i \\ \exp(\gamma_t) & se\, h_t(\mathbf{x}_i) \neq y_i \end{cases} = \frac{D_t(\mathbf{x}_i)\exp(-\gamma_t y_i h_t(\mathbf{x}_i))}{Z_t}$$

8: $t = t + 1$
9: **até** $t = T$
10: Hipótese Final:

$$H(\mathbf{x}) \leftarrow \mathrm{sgn}\left(\sum_{t=1}^{T} \gamma_t h_t(\mathbf{x})\right)$$

Figura 5. Algoritmo AdaBoost por [Chaves 2011]

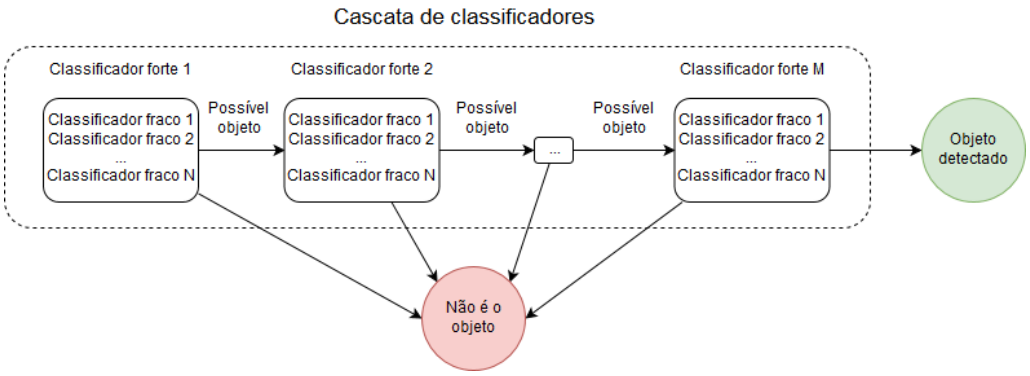

Figura 6. Ilustração de uma cascata de classificadores e como é dado o resultado de uma classificação

2.3. Base de Dados

Para a etapa de treinamento do *Haar Cascade*, separamos imagens retiradas da internet para a construção do conjunto de de dados negativos e positivos. Todas as imagens negativas obtidas foram redimensionadas para a resolução padrão de 640x480 pixels e colocadas em escala de cinza, sendo que tais imagens não continham placa de carro. As imagens positivas também foram retiradas de diversos sítios da internet e continham apenas a imagem com a região de interesse, ou seja, a placa do carro.

Figura 7. Exemplo de imagem negativa utilizada para o treinamento

Figura 8. Exemplos de imagens positivas utilizadas para o treinamento do classificador

O conjunto de imagens utilizadas para a avaliação da função haar treinada é obtida em [Laroca et al. 2018], obtido em [Dataset 2019], no qual consiste em 1500 imagens, divididas em: imagens de placas de carro cinza, imagens de veículos com placas vermelhas e imagens de moto com placa cinza, onde utilizamos apenas as placas cinzas de automóveis, pois o objetivo é utilizar placas de carros. As imagens possuem resolução de 1920x1080 com informações de onde está a região do carro, a marca do carro, a região da placa, os caracteres da placa e qual é a placa do veículo. Utilizamos a informação da região do automóvel para extrairmos e trabalharmos apenas em cima disto. Pegamos a região onde o carro está e a informação da placa, que é chamado de *ground truth*, e dentro dela aplicamos nossa metodologia para a detecção da região de placa e avaliação do classificador.

2.4. Implementação

O algoritmo foi desenvolvido usando a linguagem Python, na versão 3.6.1 e a biblioteca de código aberto OpenCV, na versão 3.4.2. Para a criação dos resultados, a base de imagens foi dividida em dois grupos: o de treinamento, com 600 imagens positivas e negativas,

Figura 9. Exemplos de imagens utilizadas para a avaliação do classificador

cerca de 20% do total de imagens. Neste estágio utilizamos a função nativa do OpenCV *opencv traincascade* para treinarmos o classificador; de teste, com 2282 imagens positivas e 600 negativas. Após o treinamento, cada imagem de teste é submetida ao algoritmo que, por sua vez, devolve a posição da placa encontrada em coordenadas de imagem (linha, coluna) ou zero caso contrário. Implementação e seu código fonte encontram-se em [Knowrafa 2019], onde estão também outros classificadores treinados, imagens positivas e negativas utilizadas para o treino e validação do método.

3. Resultados

Nessa seção, serão analisados os resultados obtidos como mostra a Figura 10. dos vários testes realizados colocando a metodologia criada em prática. Na subseção a seguir serão expostas as formas de avaliação utilizadas na pesquisa e alguns dados numéricos para uma melhor análise do classificador.

Figura 10. Imagens exemplo da classificação. Em verde o *ground truth* e em azul a região encontrada pelo classificador

3.1. Metodologia de Avaliação

Matriz de confusão é uma tabela que mostra a quantidade de classificações feitas no seu modelo, dividida em quatro principais métricas: verdadeiro positivo, falso positivo, verdadeiro negativo e falso negativo.

Verdadeiro Positivo (VP): é contabilizado quando seu modelo previu corretamente a classe a ser prevista. No nosso exemplo, é quando a região encontrada pelo classificador e a região da placa- *ground truth*- são a mesma.

Falso Positivo (FP): é contabilizado quando o modelo previu como real o que não era. No nosso exemplo, quando não existe placa na imagem e meu classificador retorna como se tivesse achado alguma região de placa.

Verdadeiro Negativo (VN): é contabilizado quando o modelo previu corretamente o que realmente não era. No nosso exemplo, é quando na imagem não existe placa de carro e o classificador retorna que não há regiões com placa de carro.

Falso Negativo (FN): é contabilizado quando o modelo previu como falsa uma amostra que era de interesse. No nosso exemplo, quando a imagem contém a região de uma placa, mas o classificador retorna como se não houvesse nenhuma região de placa. Na qual é obtida a seguinte tabela:

Tabela 1. Padrão da matriz de confusão

		Valores Preditos	
		Placas	Não Placas
Valores Reais	Placas	VP	FN
	Não Placas	FP	VN

A partir da matriz de confusão preenchida com o resultados do classificador, possuímos algumas métricas de aprendizagem de máquina que nos permite avaliar como o modelo se comporta. São elas: **Acurácia**, **Precisão**, ***Recall*** e ***f-score***.
Acurácia: é uma métrica para a avaliação de modelos de classificação. Acurácia é a fração de predições que nosso modelo acertou (Verdadeiros Positivos e Verdadeiros Negativos) sobre a quantidade total de predições. Dada pela fórmula:

$$Acurácia = \frac{VP + VN}{VP + FP + FN + VN} \tag{3}$$

Precisão: Tenta responder a seguinte questão: Que proporção de identificações positivas foi realmente correta, baseando-se apenas nas predições corretas? Dada por:

$$Precisão = \frac{VP}{VP + VN} \tag{4}$$

Recall: Tenta responder a seguinte questão: O quão bom meu modelo é para prever positivos, baseando-se apenas nas amostras positivas? Dada por:

$$Recall = \frac{VP}{VP + FN} \tag{5}$$

f-score: O f-score nos mostra o balanço entre a precisão e o recall de nosso modelo.

$$f\text{-}score = 2 * \frac{Precisão * Recall}{Precisão + Recall} \tag{6}$$

No nosso modelo de avaliação, uma classificação é considerada verdadeiro positiva quando a diferença entre o centro do *ground truth* e a região encontrada é menor que uma certa quantidade de pixels. Também consideramos se a região encontrada está apenas na parte de baixo da imagem. Nos gráficos a seguir, mostramos como as métricas se comportam quando a diferença entre os centros é modificada.

Tabela 2. Tabela das médias das matrizes de confusão

		Valores Preditos	
		Placas	Não Placas
Valores Reais	Placas	1528	754
	Não Placas	201	399

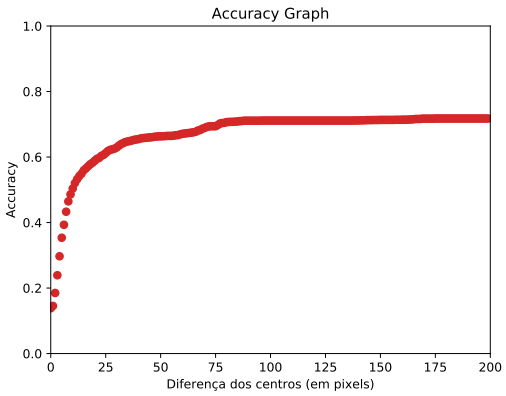

Figura 11. Gráfico da Acurácia: diferença entre o centro da região encontrada pelo classificador

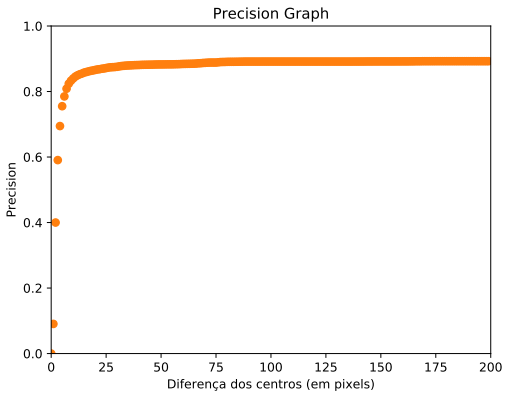

Figura 12. Gráfico da Precisão: diferença entre o centro da região encontrada pelo classificador

4. Interpretação dos Resultados

Pode-se observar pela Figura 11, que a à medida que a diferença entre os centros da região encontrada pelo algoritmo e o *ground truth* aumenta, a acurácia também aumenta devido a uma maior flexibilidade na localização da janela. Observa-se também que, após uma diferença de aproximadamente 75 pixels, há um aumento pouco significativo na acurácia. A precisão, na Figura 12, apresenta um comportamento mais estável em função da diferença de centros, ficando aproximadamente constante em 83% após uma diferença aproximada de 20 pixels. A curva de *Recall*, apresentada na Figura 13, assemelha-se à curva da acurácia, sendo que a partir de aproximadamente 75 pixels de diferença do centro ela se mantém com pouca variação. Pela curva *f-score*, na Figura 14, que representa o balanço entre o *recall* e a precisão, pode-se observar que a partir de aproximadamente 75 pixels de diferença os valores apresentam pouco ganho.

5. Conclusão

Este trabalho teve por objetivo a construção de um método de detecção de placas de licenciamento em imagens usando características de Haar e AdaBoost. Como exposto

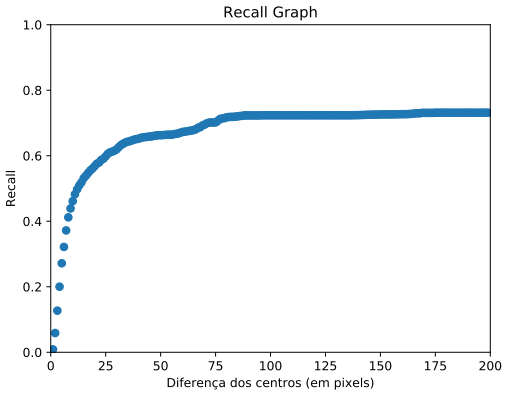

Figura 13. Gráfico do *Recall*: diferença entre o centro da região encontrada pelo classificador

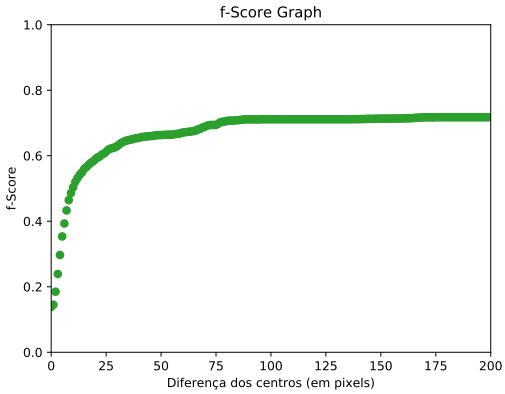

Figura 14. Gráfico do f-score: diferença entre o centro da região encontrada pelo classificador

nos gráficos da seção de resultados, o método proposto neste trabalho teve êxito em seu objetivo, com o aumento da acurácia à medida que a distância entre os centros da regiões detectadas e o *ground truth* permite uma maior flexibilização.

Observou-se uma estabilização dos valores de acurácia, precisão, *recall* e *f-score* permaneceram estáveis a partir de uma distância de 75 pixels entre a posição esperada da placa com a posição encontrada pelo algoritmo. Isso indica que o método consegue detectar as regiões de placa mas não possui boa precisão em relação à posição do padrão esperado.

Apesar do bom resultado, a expectativa dos pesquisadores em relação à taxa detecção de placas era maior do que a obtida, visto que foi utilizado uma quantidade razoável de amostras durante a fase de treinamento da função Haar. Porém, o resultado obtido é compreensível visto que o custo computacional do treinamento foi um fator limitante, tanto em relação a tempo de processamento quando para a quantidade de memória. Para características de Haar de maior dimensão, a execução do algoritmo de treinamento tornou-se inviável para os recurso computacional disponível para esta pesquisa, havendo duração aproximada de 3 dias para cada treinamento/teste do sistema. Sendo assim, foi necessário escolher janelas menores ou interromper o treinamento antes de ser finalizado completamente.

Referências

Ahn, C., Lee, B., Yang, S., and Park, S. (2017). Design of car license plate area detection algorithm for enhanced recognition plate. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pages 1–4.

AlyanNezhadi, M. M., Hashemi, S. M. R., and Abolghasemi, V. (2017). License plate detection in complex scenes based on fusion of gaussian filtering and bayesian network. In *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pages 0022–0026.

Chandra, H., Michael, Hadisaputra, K. R., Santoso, H., and Anggadjaja, E. (2017). Smart parking management system: An integration of rfid, alpr, and wsn. In *2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS)*, pages 1–6.

Chaves, B. B. (2011). Estudo do algoritmo adaboost de aprendizagem de máquina aplicado a sensores e sistemas embarcados. Master's thesis, São Paulo: Universidade de São Paulo, Escola Politécnica.

Dataset, A. (2019). Repository containing image database used in the training phase. `https://web.inf.ufpr.br/vri/databases/ufpr-alpr/`. accessed 2019-09-23.

Du, S., Ibrahim, M., Shehata, M., and Badawy, W. (2013). Automatic license plate recognition (alpr): A state-of-the-art review. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 23, pages 311–325.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. In *Journal of Computer and System Sciences*, volume 55, pages 119–139.

Gibson, W. (2005). *Pattern recognition*. Berkley Books, 1st edition.

Knowrafa (2019). Repository containing source code and image database. `https://github.com/knowrafa/ic-plate-recognition`. accessed 2019-09-23.

Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R., and Menotti, D. (2018). A robust real-time automatic license plate recognition based on the yolo detector. pages 1–10. IEEE.

Quiros, A. R. F., Bedruz, R. A., Uy, A. C., Abad, A., Bandala, A., Dadios, E. P., Maningo, J. M., and Vicerra, R. R. (2017). Localization of license plates using optimized edge and contour detection technique. In *TENCON 2017 - 2017 IEEE Region 10 Conference*, pages 1075–1080.

Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. Springer, 1st edition.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I.

Wang, A., Liu, X., Han, Y., and Qi, C. (2012). License plate location algorithm based on edge detection and morphology. In *2012 7th International Forum on Strategic Technology (IFOST)*, pages 1–4.