Classificação de Vagas de Estacionamento usando Aprendizagem de Máquina

Gabriel Rosa e Silva e Walter Antonio Gontijo

***Resumo*— Este artigo apresenta contribuições para a classificação de vagas de estacionamento monitoradas por uma câmera de vídeo, usando aprendizagem de máquina com classificação por *Support Vector Machine* (SVM). As contribuições apresentadas incluem a utilização de uma aproximação de um *kernel* do tipo *Radial Basis Function* (RBF), e de um *threshold* na região de decisão. Tais contribuições permitem aumentar o desempenho do classificador sem um impacto significativo no tempo de execução em sistemas embarcados. O método proposto é implementado em um sistema embarcado, classificando até 256 vagas por imagem com o desempenho esperado.**

***Palavras-Chave*— Aprendizagem de máquina, SVM, RBF, sistemas embarcados**

***Abstract*— In this paper we present contributions to the classification of parking spaces monitored by video cameras, using machine learning with classification by Support Vector Machine. The contributions include the usage of a Radial Basis Function kernel and of a threshold in the decision region. Such contributions lead to better classification accuracy with little impact in the performance of an embedded implementation. The proposed method is implemented in an embedded environment, classifying up to 256 parking spaces per image with the expected accuracy.**

***Keywords*— Machine learning, SVM, RBF, embedded systems**

I. Introdução

Atualmente, na maioria das cidades, encontrar uma vaga de estacionamento é um problema considerável. Estacionamentos monitorados em grandes centros urbanos mostram ocupação próxima de 100% durante a maior parte do horário comercial [1]. Estima-se ainda que, em média, 30% dos carros congestionados em regiões centrais desses centros estão à procura de vaga para estacionar [2].

Historicamente, tentou-se solucionar este problema através de sistemas automatizados com sensores de presença de veículos. Os sensores identificam vagas disponíveis e um sistema centralizado faz a aquisição desses dados para repassar a informação ao usuário. Estudos recentes apontam a viabilidade econômica de tal sistema, mas custos com sensor físico, cabeamento e manutenção para cada vaga são um obstáculo a implementações em larga escala [3].

Com a popularização de plataformas que seguem o paradigma Internet das Coisas (IoT), tem-se um crescimento na oferta de serviços que visam automatizar aspectos da vida moderna [4]. A disponibilidade de plataformas embarcadas e sensores cada vez mais eficientes e econômicos tem permitido a aplicação desse paradigma, por exemplo, na automação de serviços de utilidade pública em *Smart Cities* [5]. Uma arquitetura comum nesse tipo de aplicação utiliza uma *Wireless Sensor Network* (WSN), com sensores distribuídos em uma área geográfica se comunicando através de tecnologias sem fio [6]. Em aplicações onde são utilizadas câmeras de vídeo, essa rede recebe o nome de *Visual Sensor Network* (VSN) [7].

A aplicação de uma VSN em conjunto com métodos de classificação por aprendizagem de máquina é uma alternativa ao uso de sensores de presença na identificação de vagas de estacionamento [8]. Essa estratégia se mostra interessante, uma vez que cada câmera pode capturar informações de um grande número de vagas. Adicionalmente, muitos estacionamentos já possuem câmeras por motivos de segurança, reduzindo custos de implementação.

Tradicionalmente, um sistema deste porte tem a imagem capturada pelas câmeras e transmitida integralmente a um servidor centralizado que classifica o estado das vagas. A este paradigma é dado o nome de *compress-then-analyze* (CTA). O uso de métodos de aprendizagem de máquina permite que as imagens sejam classificadas em plataformas embarcadas e apenas o estado (vaga ocupada ou livre) seja transmitido a um servidor. Esse paradigma é conhecido como *analyze-then-compress* (ATC) e se mostra interessante para aplicações em *Smart Cities* por seu baixo custo e alta escalabilidade [9].

Em [10], é apresentado um método de classificação com aprendizagem de máquina que utiliza uma *Support Vector Machine* (SVM) alimentada com informações de textura da imagem. Já em [11], é utilizada uma SVM alimentada com os tons de cor da imagem, obtidos com menor custo computacional. Os ganhos de performance nesse segundo método permitem a utilização de uma arquitetura organizada em VSNs, porém com maior taxa de erros de classificação.

Neste artigo, são mostradas melhorias no método apresentado em [11], visando diminuir as taxas de erro de classificação sem impactar significativamente a performance. A estratégia de classificação utilizada em [11] (SVM alimentada com os tons de cor da imagem) é mantida neste artigo, por seu custo computacional comprovadamente baixo. As melhorias apresentadas têm seu desempenho validado em uma implementação embarcada.

Este artigo é organizado como se segue: Na Seção II, são apresentadas as contribuições propostas. Na Seção III, são apresentados os resultados obtidos na simulação e na validação. Finalmente, na Seção IV, são apresentadas as conclusões do artigo.

Gabriel Rosa e Silva e Walter Antonio Gontijo¸ LINSE – Laboratório de Circuitos e Processamento de Sinais, Departamento de Engenharia Elétrica e Eletrônica, Universidade Federal de Santa Catarina (UFSC), Florianópolis, SC, Brasil, e-mails: grsilva@linse.ufsc.br e walter@linse.ufsc.br.

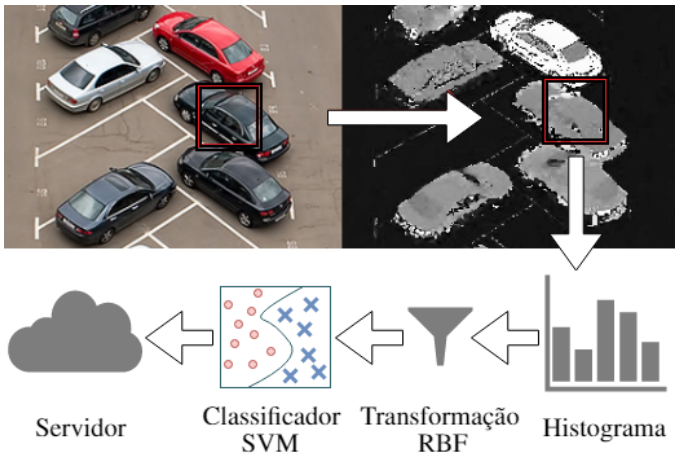

Fig. 1. Uma região quadrada no centro da vaga é extraída e tem seus *pixels* convertidos em tons de cor. A partir dos tons, é gerado um histograma que passa por uma transformação por *kernel* RBF antes de entrar no classificador SVM. A informação de classificação da vaga é enviada a um servidor.

II. Contribuições Propostas

A estratégia proposta apresenta duas modificações na classificação por SVM: a introdução de uma aproximação de um *kernel* RBF e a adição de um *threshold* na região de decisão.

A utilização de um *kernel* permite compensar possíveis não-linearidades do conjunto de dados, aumentando assim o desempenho do classificador. Por seu baixo número de parâmetros e boa estabilidade numérica, um *kernel* RBF é o mais recomendado para a maioria das aplicações [12]. Uma aproximação de um *kernel* RBF é utilizada por motivos de performance [13].

A segunda modificação consiste em um *threshold* na região de decisão para indicar quando uma classificação não é confiável.

A. Classificação

Deseja-se classificar se uma vaga de um estacionamento monitorado por uma câmera de vídeo está ou não ocupada. Conforme Fig. 1, a região de análise é delimitada por um quadrado no centro da vaga. Inicialmente, é realizada a conversão do espaço de cor RGB para HSV, permitindo a extração dos tons de cor (*hue*). A partir dos tons, é calculado o histograma normalizado com N coeficientes, que é armazenado no vetor $\boldsymbol{x}_{in}$. Esse vetor será a entrada não transformada do sistema.

O histograma normalizado em $\boldsymbol{x}_{in}$ é então transformado por uma aproximação de um *kernel* RBF, passando a ter um número maior de coeficientes. A equação de transformação é dada por

$$\boldsymbol{x}_t = \sqrt{\frac{2}{M}} cos(\boldsymbol{R} \times \boldsymbol{x}_{in} + \boldsymbol{r}) \tag{1}$$

onde $M > N$ é o número de coeficientes do vetor transformado, $\boldsymbol{R}$ é a matriz de transformação de tamanho $M \times N$ e $\boldsymbol{r}$, o vetor de *offset* com M coeficientes. O vetor transformado é denotado por $\boldsymbol{x}_t$. Os vetores $\boldsymbol{x}_{in}$, $\boldsymbol{r}$ e $\boldsymbol{x}_t$ são vetores coluna.

A partir do vetor transformado, um classificador SVM avalia a existência ou não de veículo na vaga. Um modelo de classificação obtido durante um processo de treinamento é utilizado na realização desta avaliação, que é dada por

$$d = \boldsymbol{x}_t \cdot \boldsymbol{w} + b \tag{2}$$

onde o modelo é representado pelo vetor $\boldsymbol{w}$ de M coeficientes e *offset* b, que podem ser interpretados respectivamente como o vetor normal e a distância da origem de um plano que separa as duas classificações. A equação resulta no escalar d, que representa a distância do vetor $\boldsymbol{x}_t$ ao plano definido pelo modelo. O sinal de d indica a classe da vaga analisada e o seu módulo indica a confiabilidade do resultado.

Um passo adicional para remover amostras não confiáveis é realizado através da comparação do valor de d com um *threshold*

$$C = \begin{cases} +1, & \text{se } d \geq K \\ -1, & \text{se } d \leq -K \\ 0, & \text{se } -K < d < K \end{cases} \tag{3}$$

onde K representa o *threshold* e C é um inteiro que representa a classe obtida: (-1) se a vaga estiver vazia, $(+1)$ se a vaga estiver ocupada e (0) se o resultado não for confiável.

B. Implementação

O modelo de classificação foi gerado em um processo de treinamento com um banco de mais de 600 mil imagens de vagas individuais pré-classificadas. As imagens foram capturadas por câmeras *full HD* de três ângulos difierentes de dois estacionamentos e armazenadas em amostras de formato JPEG com 100% de qualidade e resolução de 1280x720 *pixels*. O banco é composto por três *datasets*, representando cada um dos ângulos de captura, sendo eles: décimo andar do prédio administrativo da Pontifícia Universidade Católica do Paraná (PUC), quarto e quinto andares da Universidade Federal do Paraná (UFPR04 e UFPR05, respectivamente). Cada *dataset* é separado ainda de acordo com as condições climáticas no momento da captura (ensolarado, nublado, chuvoso) [14].

Um processo de validação do modelo de treinamento foi realizado para avaliar a taxa de precisão de suas classificações. Para garantir uma avaliação imparcial, os *datasets* foram separados em partes iguais, sendo o treinamento realizado com uma metade e o modelo resultante validado com a outra. A distribuição das amostras segue a Tabela I.

TABELA I
Amostras

Dataset	Treinamento	Validação	Total
PUC	200.579	200.579	401.158
UFPR04	52.881	52.881	105.762
UFPR05	82.850	82.850	165.700
Total	336.310	336.310	672.620

Antes de realizar o processo de treinamento, cada amostra foi processada para extrair um histograma dos seus tons de cor

com número de coeficientes $N = 180$, empacotado em lotes de arquivos binários.

O treinamento do modelo foi realizado utilizando a biblioteca *scikit-learn*, em Python [15]. Devido ao grande número de amostras, foi utilizado um método de *Stochastic Gradient Descent* (SGD), possibilitando o treinamento em lotes de amostras [8]. O tamanho dos lotes utilizados foi de 50 mil amostras.

Junto com o treinamento, foi realizado o processo de geração do *kernel* RBF aproximado, utilizando funções da biblioteca *scikit-learn* baseadas no método *Random Kitchen Sinks* [16].

Um script em Python foi criado para obter métricas mais detalhadas durante a validação. Nesse script, foram implementadas ferramentas para avaliar modelos de diferentes parâmetros através de uma *Grid Search* [12]. O parâmetro de *threshold* K foi escolhido através de uma biseção, até que uma determinada porcentagem das amostras menos confiáveis fossem descartadas.

Os parâmetros que resultaram na melhor precisão foram: parâmetro de regularização do método SGD $\lambda = 10^{-5}$, parâmetro $\gamma = 13$ no *kernel* e threshold $K = 0,845$, descartando-se 10% das amostras menos confiáveis do banco.

III. Resultados

A. Avaliação do Método

O modelo foi treinado e validado com as amostras dos três *datasets* (ver Tabela I), utilizando valores de $M = \{300, 400, 500, 600, 700, 800, 900, 1000$ e $1200\}$ coeficientes. O método apresentado em [11] também foi reproduzido, com modelo de 180 coeficientes, sem transformação por *kernel* RBF e com *threshold* $K = 0$ (ou seja, sem *threshold*). As precisões obitdas no processo de validação são apresentadas na Fig. 2.

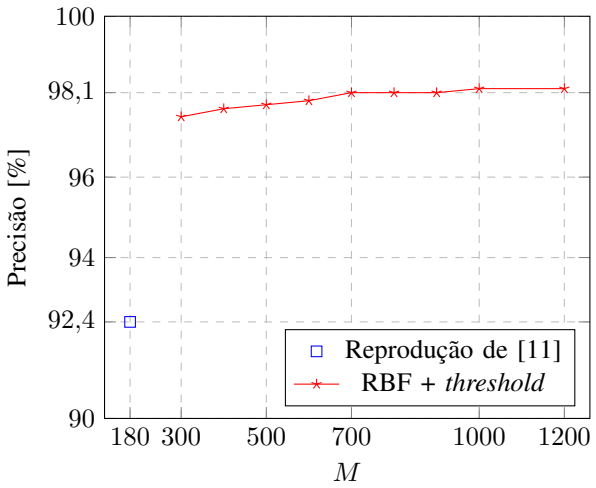

Fig. 2. Comparação da precisão de classificação para diferentes valores de M.

Observa-se na Fig. 2 que a precisão aumenta conforme cresce o valor de M. Porém, como esse aumento de precisão possui custo associado, foi fixado o tamanho $M = 700$ para obter uma precisão de classificação de 98,1%, bem acima dos 92,4% obtidos na reprodução de [11] ($M = 180$).

Foram realizadas ainda validações de 9 subconjuntos do banco, representando as diferentes condições climáticas para cada um dos três *datasets*, com os resultados apresentados na Tabela II.

TABELA II
AVALIAÇÃO SOB DIFERENTES CONDIÇÕES CLIMÁTICAS (PRECISÃO EM %)

	Ensolarado	Nublado	Chuvoso	Total
PUC	98,20	99,74	99,49	98,99
UFPR04	96,25	98,22	97,95	97,13
UFPR05	96,45	98,37	96,87	97,15

As condições climáticas interferem na classificação, com a maior precisão obtida em dias nublados, possivelmente por causa da iluminação difusa.

B. Custo Computacional

O procedimento de *thresholding* tem custo computacional irrisório, portanto foi considerado que o custo computacional estará associado unicamente à transformação RBF. Com os valores utilizados de $M = 700$ e $N = 180$, o custo de memória associado à matriz e ao vetor de transformação ($\boldsymbol{R}$ e $\boldsymbol{r}$, respectivamente) é de $M(N + 1)$ números de ponto flutuante. Na representação de ponto flutuante *single precision* de 32 bits, o espaço ocupado em memória é de cerca de 495 kB. O custo computacional da transformação é analisado, uma vez que essa é aplicada sempre que uma vaga deve ser classificada.

TABELA III
OPERAÇÕES EM PONTO FLUTUANTE PARA $M = 700$ E $N = 180$

Tipo	Quantidade
mult	126.000
add	127.400
cos	700
Total	254.100

Uma estimativa do custo de operações em ponto flutuante é apresentada na Tabela III. Essa estimativa não considera possíveis otimizações do código. Mesmo sem otimizações, o número total de operações está bem abaixo da *performance* observada em placas modernas utilizadas em aplicações IoT, conforme [17].

C. Implementação Embarcada

Os resultados obtidos na validação em simulação motivaram uma implementação do método em um sistema embarcado. A plataforma utilizada foi a Intel Galileo Gen 2 [18], com processador Intel Quark X1000 32-bit 400 MHz e 256 MB de memória RAM, rodando o sistema Yocto Linux [19].

A classificação foi realizada a partir de imagens armazenadas na memória da placa. A partir de um script em Python, cada imagem de teste foi segmentada, transformada de RGB para HSV e gerado o histograma dos tons. Esse histograma foi classificado de acordo com o modelo apresentado. A saída

do classificador para todas as vagas foi concatenada em uma sequência de bits, que foi enviada via HTTP para um banco de dados na nuvem.

TABELA IV
COMPARAÇÃO DO TEMPO DE EXECUÇÃO ENTRE OS MÉTODOS DE REFERÊNCIA E PROPOSTO

Vagas	Tempo Ref. (s)	Tempo Prop. (s)
25	20	20
100	59	61
256	120	127

O tempo total de execução, incluindo o carregamento da imagem na memória e a comunicação com o servidor, foi avaliado para configurações com números de vaga igual a $\{25$, 100 e $256\}$. Os resultados são apresentados na Tabela IV. O resultado obtido para a classificação de um estacionamento com 100 vagas foi de 61s. Nesse caso, a precisão de classificação obtida foi de 98% de acertos, conforme esperado. O método de referência com $M = 180$ também foi avaliado, classificando 100 vagas em 59s.

Foram avaliados também os tempos de execução dos procedimentos de classificação e de processamento (segmentação da imagem, transformação de RGB pra tom e cálculo do histograma), para diferentes valores de M (180, 300, 400, 500, 600, 700, 800, 900, 1000 e 1200). Utilizou-se nessa avaliação a configuração com maior número de vagas consideradas (256). Para $M = 180$ não é realizada transformação RBF, conforme [11]. Os resultados são apresentados na Tabela V. Como previsto, o tempo de classificação está associado à transformação RBF, sendo próximo a zero na reprodução do método de referência e crescente com o aumento de M. Observa-se na Tabela V que a diferença no tempo de processamento + classificação entre o método apresentado ($M = 700$) e o de referência ($M = 180$) é de 7s, representando um aumento de apenas 5,97%.

TABELA V
TEMPO DE CLASSIFICAÇÃO PARA DIFERENTES VALORES DE M.

M	Classificação (s)	Processamento (s)	Class. + Proc. (s)
180	0,128	117,16	117,29
300	2,96	117,50	120,46
400	5,49	117,53	123,02
500	4,87	117,99	122,86
600	6,24	117,25	123,49
700	6,75	117,54	124,29
800	10,58	117,61	128,19
900	8,72	117,39	126,11
1000	11,41	117,90	129,31
1200	16,48	117,29	133,77

IV. CONCLUSÕES

Neste artigo, são apresentadas contribuições para aumentar o desempenho na classificação de vagas de estacionamento de veículos a partir de imagens de câmeras de vídeo. O método considerado como referência utiliza um classificador SVM linear alimentado com os tons de cor da imagem, possui um desempenho de 92,1% e pode ser executado em um sistema embarcado. As contribuições propostas incluem a utilização de uma aproximação de um *kernel* RBF e de um *threshold* na região de decisão. Tais contribuições elevam o desempenho do classificador para 98,2% com impacto de 5,97% no tempo de execução do algoritmo original. O método proposto foi implementado em um sistema embarcado que reproduziu em tempo real o desempenho obtido em simulação.

REFERÊNCIAS

[1] S. F. M. T. Authority, "Sfpark: Putting theory into practice," Aug. 2011.

[2] D. C. Shoup, "Cruising for parking," *Transport Policy*, vol. 13, no. 6, pp. 479–486, Nov. 2006.

[3] M. Tahon, S. Verbrugge, B. Lannoo, J. Van Ooteghem, P. De Mil, D. Colle, M. Pickavet, and P. Demeester, "Parking sensor network: Economic feasibility study of parking sensors in a city environment is well," in Proceedings of *9th Conference on Telecommunications Internet and Media Techno Economics (CTTE)*, pp. 1–8, Jun. 2010.

[4] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International Journal of Communication Systems*, vol. 25, no. 9, p. 1101, Sep. 2012.

[5] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, and H. J. Scholl, "Understanding smart cities: An integrative framework," in Proceedings of *45th Hawaii International Conference on System Science (HICSS)*, pp. 2289–2297, Jan. 2012.

[6] C. Guy, "Wireless sensor networks," in Proceedings of *IEEE 6th International Symposium on Instrumentation and Control Technology: Signal Analysis, Measurement Theory, Photo-Electronic technology, and Artificial Intelligence*, p. 63571I, Nov. 2006.

[7] D. Kundur, C.-Y. Lin, and C.-S. Lu, "Visual sensor networks," *EURASIP Journal on Advances in Signal Processing*, Article no. 021515, Mar. 2007.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ: Springer-Verlag, 2006.

[9] A. Redondi, L. Baroffio, M. Cesana, and M. Tagliasacchi, "Compress-then-analyze vs. analyze-then-compress: Two paradigms for image analysis in visual sensor networks," in Proceedings of *IEEE 15th International Workshop on Multimedia Signal Processing (MMSP)*, pp. 278–282, Sep. 2013.

[10] P. Almeida, L. S. Oliveira, E. Silva, A. Britto, and A. Koerich, "Parking space detection using textural descriptors," in Proceedings of *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3603–3608, Oct. 2013.

[11] L. Baroffio, L. Bondi, M. Cesana, A. E. Redondi, and M. Tagliasacchi, "A visual sensor network for parking lot occupancy detection in smart cities," in Proceedings of *IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 745–750, Dec. 2015.

[12] C.-W. Hsu, C.-C. Chang, C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[13] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," in Proceedings of *Advances in Neural Information Processing Systems 20 (NIPS)*, pp. 1177–1184, 2008.

[14] P. R. De Almeida, L. S. Oliveira, A. S. Britto, E. J. Silva, and A. L. Koerich, "Pklot–a robust dataset for parking lot classification," *Expert Systems with Applications*, vol. 42, no. 11, pp. 4937–4949, Jul. 2015.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011.

[16] A. Rahimi and B. Recht, "Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning," in Proceedings of *Advances in Neural Information Processing Systems 21 (NIPS)*, pp. 1313–1320, 2009.

[17] K. Keville, "Intel bay trail nuc kit ubuntu linux comparison," [Accessed 3-April-2017]. [Online]. Available: https://openbenchmarking.org/result/1402191-SO-1402118PL06

[18] M. C. Ramon, *Intel Galileo and Intel Galileo Gen 2*. Berkeley, CA: Apress, Dec. 2014.

[19] Y. Project and L. Foundation, "Yocto project," [Accessed 3-April-2017]. [Online]. Available: https://www.yoctoproject.org/