Eduardo Pimentel de Alvarenga

Identificação de Caracteres para Reconhecimento Automático de Placas Veiculares

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA
Programa de Pós–Graduação em Informática

Rio de Janeiro
Abril de 2014

Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Pimentel de Alvarenga

Identificação de Caracteres para Reconhecimento Automático de Placas Veiculares

Dissertação de Mestrado

Dissertação apresentada ao Programa de Pós–Graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Ruy Luiz Milidiu

Rio de Janeiro
Abril de 2014

Pontifícia Universidade Católica
do Rio de Janeiro

Eduardo Pimentel de Alvarenga

Identificação de Caracteres para Reconhecimento Automático de Placas Veiculares

Dissertação apresentada ao Programa de Pós–Graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio como requisito parcial para obtenção do grau de Mestre em Informática. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Ruy Luiz Milidiu
Orientador
Departamento de Informática — PUC–Rio

Prof. Marco Antonio Casanova
PUC-Rio

Prof. Bruno Feijó
PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico — PUC–Rio

Rio de Janeiro, 14 de Abril de 2014

Eduardo Pimentel de Alvarenga

Graduou–se Bacharel em informática pela PUC Rio (PUC-Rio/RJ). Atua como analista de sistemas na área de pesquisa e desenvolvimento há 14 anos, com foco em criptografia, segurança e otimização de sistemas em ambientes de recursos limitados.

Ficha Catalográfica

Alvarenga, Eduardo Pimentel de

Identificação de Caracteres para Reconhecimento Automático de Placas Veiculares / Eduardo Pimentel de Alvarenga; orientador: Ruy Luiz Milidiu. — 2014.

50 f. : il. (color); 30 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2014.

Inclui bibliografia.

1. Informática – Teses. 2. Reconhecimento Automático de Placas (ALPR). 3. Reconhecimento Ótico de Caracteres (OCR). 4. Perceptron. 5. Aprendizado de Máquinas. 6. Geração de Atributos Guiada por Entropia (EFG). I. Milidiu, Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Para os meus pais e avós, que sempre incentivaram meus estudos, apontando o caminho e abrindo as portas necessárias. Therezinha e Mauricio, Nina e Henrique, Mãe e Pai: mesmo que procurasse no mundo inteiro, não conseguiria montar um time (e torcida!) melhor do que este.

Em memória de minha avó Nina, que nos deixou na reta final, mas com certeza continua no time, torcendo um pouco mais de longe.

Para minha mais que fantástica esposa Gisele, por não me deixar perder o foco, me ajudando mais do que ela mesma pode imaginar. Procuraria por você no mundo todo, mas sei que sempre a encontrarei ao meu lado.

E para as minhas irmãs, Marcella e Gabriela, amigas de uma vida e meia, com quem compartilho sonhos e metas, anseios e preocupações.

Esta etapa está terminando, conto com vocês para as próximas!

Agradecimentos

À minha família, pelo carinho e incentivos constantes, sempre presente nos momentos importantes.

À Doutora Carmen, minha mãe e revisora, e ao meu pai, pela motivação, inspiração e preocupação permanentes durante todo o curso.

À minha esposa Gisele, que trabalhou tanto para que tudo desse certo, pela companhia nas noites de sono perdido, pelas palavras de conforto nos dias de desespero e pelo amor incondicional que sempre demonstrou.

Às minhas irmãs, afilhada e sobrinho, pela alegria que trazem, mesmo quando não nos vemos com a frequência que gostaríamos.

Ao Professor Ruy Milidiu, pelo aprendizado desde o primeiro período, por me orientar neste trabalho e pelo convite a participar do LEARN, criando novas oportunidades de aprendizado neste breve convívio.

Aos velhos e novos amigos pela compreensão nas festas perdidas, encontros adiados e passeios cancelados. Acreditem, valeu a pena! Em especial aos amigos encontrados no mestrado, obrigado pela companhia durante esta caminhada. Não teria sido o mesmo sem vocês.

À Carol Valadares, parceira de estudos e trabalhos desde o primeiro período, pela ajuda com as peculiaridades acadêmicas e por compartilhar as angústias e alegrias desde o começo do curso.

À Montreal Informática, em especial ao meu chefe Luiz Antônio, por viabilizar esta oportunidade única.

Aos professores e funcionários do Departamento de Informática da PUC-Rio, por mais esta oportunidade de aprendizado. E à PUC-Rio, pelo apoio financeiro.

Resumo

Alvarenga, Eduardo Pimentel de; Milidiu, Ruy Luiz. **Identificação de Caracteres para Reconhecimento Automático de Placas Veiculares**. Rio de Janeiro, 2014. 50p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Sistemas de reconhecimento automático de placas (ALPR na sigla em inglês) são geralmente utilizados em aplicações como controle de tráfego, estacionamento, monitoração de faixas exclusivas entre outras aplicações. A estrutura básica de um sistema ALPR pode ser dividida em quatro etapas principais: aquisição da imagem, localização da placa em uma foto ou frame de vídeo; segmentação dos caracteres que compõe a placa; e reconhecimento destes caracteres. Neste trabalho focamos somente na etapa de reconhecimento. Para esta tarefa, utilizamos um Perceptron multiclasse, aprimorado pela técnica de geração de atributos baseada em entropia. Mostramos que é possível atingir resultados comparáveis com o estado da arte, com uma arquitetura leve e que permite aprendizado contínuo mesmo em equipamentos com baixo poder de processamento, tais como dispositivos móveis.

Palavras–chave

Reconhecimento Automático de Placas (ALPR); Reconhecimento Ótico de Caracteres (OCR); Perceptron; Aprendizado de Máquinas; Geração de Atributos Guiada por Entropia (EFG).

Abstract

Alvarenga, Eduardo Pimentel de; Milidiu, Ruy Luiz (Advisor). **Optical Character Recognition for Automated License Plate Recognition Systems**. Rio de Janeiro, 2014. 50p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

ALPR systems are commonly used in applications such as traffic control, parking ticketing, exclusive lane monitoring and others. The basic structure of an ALPR system can be divided in four major steps: image acquisition, license plate localization in a picture or movie frame; character segmentation; and character recognition. In this work we'll focus solely on the recognition step. For this task, we used a multiclass Perceptron, enhanced by an entropy guided feature generation technique. We'll show that it's possible to achieve results on par with the state of the art solution, with a lightweight architecture that allows continuous learning, even on low processing power machines, such as mobile devices.

Keywords

Automatic License Plate Recognition (ALPR); Optical Character Recognition (OCR); Perceptron; Machine Learning; Entropy Guided Feature Generation (EFG).

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

Visão Computacional é o campo que estuda o processamento e análise de imagens, com o intuito de interpretar as informações contidas nelas. Uma forma de realizar esta análise é baseada na teoria do aprendizado de máquinas supervisionado, onde um conjunto de imagens anotadas - denominado *dataset* de treino - é usada como entrada para treinar modelos de aprendizado de máquina. Estes podem ser usados posteriormente para classificar uma imagem desconhecida em uma das categorias determinadas durante a etapa de aprendizado, obtendo, portanto, informação significativa a respeito do conteúdo representado na imagem.

Com a popularização de *smart devices*, como celulares e tablets, torna-se cada vez mais interessante integrar a tecnologia de visão computacional com a mobilidade oferecida por estes dispositivos. Apesar da constante evolução destes equipamentos, sua capacidade de processamento ainda é limitada, seja pelo próprio processador disponível ou pelas restrições impostas pelo uso da bateria.

Os Sistemas de Reconhecimento Automatizado de Placas (ALPR) são um exemplo de aplicação da Visão Computacional. Estes sistemas têm o objetivo de reconhecer os caracteres (letras e números) que compõem uma placa veicular presente em uma imagem ou vídeo.

Um sistema ALPR pode ser divido em quatro etapas: captura da imagem, extração da placa, segmentação dos caracteres e reconhecimento dos caracteres. Em (Du et al., 2013) os autores listam as dificuldades encontradas por este tipo de sistema. Neste trabalho exploramos somente a quarta etapa, de reconhecimento dos caracteres.

1.1
Descrição do Problema

Em um sistema ALPR, a etapa de reconhecimento de caracteres óticos (OCR) consiste em analisar um fragmento da imagem contendo a representação de um único caractere e identificar qual a letra ou número está contida na imagem.

Este processo é executado repetidas vezes, sendo fundamental que a execução apresente alta performance. Além do reduzido tempo de resposta, a taxa de erro de classificação deve ser mínima, uma vez que durante o reconhecimento de uma única placa este processo será executado várias vezes - no caso específico de placas brasileiras, são sete caracteres por placa, sendo três letras e quatro números.

Além da performance durante o reconhecimento, torna-se atrativo que o processo de aprendizado do modelo seja o mais simples possível, a fim de que se possa utilizar mecanismos de aprendizado continuado nas situações em que o usuário identifique um erro no reconhecimento de uma placa.

1.2 Objetivos

O objetivo deste trabalho é apresentar uma solução para o problema de OCR que apresente resultados comparáveis aos mecanismos considerados estado da arte, oferecendo excelente desempenho no reconhecimento dos caracteres e um processo de aprendizado contínuo que pode ser executado diretamente nos dispositivos dos usuários finais.

Além disso, o trabalho também demonstra que o mecanismo de geração de atributos descrito em (Santos e Milidiú, 2012), inicialmente utilizado em aplicações de linguagem natural, também pode ser aplicado à visão computacional com sucesso.

1.3 Estrutura da Dissertação

Nesta introdução apresentamos uma visão geral do problema, dando uma breve explicação do que são sistemas de Reconhecimento Automatizado de Placas, além do escopo e objetivo do trabalho proposto.

O restante deste documento está organizado da seguinte maneira:

No Capítulo 2 é apresentado em mais detalhes a estrutura dos sistemas ALPR, descrevendo o funcionamento, objetivo e algumas técnicas utilizadas em suas etapas.

No Capítulo 3 são discutidos alguns trabalhos realizados na mesma área, abordando os pontos positivos e diferenças em relação à solução proposta.

No Capítulo 4 descreve-se a solução proposta, apresentando os mecanismos do Perceptron e de Geração de Atributos.

No Capítulo 5 é descrito o dataset utilizado e os tratamentos realizados para simular o funcionamento das etapas anteriores de um sistema ALPR.

No Capítulo 6 é apresentada a conclusão do trabalho e sugestões de pesquisas futuras que podem ser realizadas com base no que foi apresentado.

2
Reconhecimento Automático de Placas Veiculares

2.1
Captura da Imagem

A primeira e mais simples etapa de um sistema ALPR é a captura da imagem. Dependendo do propósito do sistema e do ambiente onde este será usado, pode-se utilizar diferentes tipos de equipamento para esta captura.

Em sistemas de controle de tráfego, normalmente são usadas câmeras fotográficas de alta resolução, com ou sem capacidade de captura de imagens de infravermelho para melhor resolução em condições de baixa luminosidade.

Figura 2.1: Imagem capturada por uma câmera de avanço de sinal

Em outros casos, como no caso de *smart devices*, pode ser mais interessante trabalhar com um fluxo de vídeo. Neste caso, o sistema deve selecionar um ou mais quadros do vídeo para analisar o seu conteúdo. Algumas estratégias para realizar esta seleção baseiam-se em uma análise prévia da imagem, para detectar por exemplo se existe uma placa ou se esta está obstruída ou distorcida em um quadro. Outra abordagem consiste em processar tantos quadros quanto possível, com o objetivo de obter uma classificação mais confiável através da média das classificações de cada quadro. Por fim, também é frequente o uso de técnicas de comparação de quadros para reprocessar somente aqueles que

possuem diferenças significativas, que potencialmente levariam a resultados diferentes dos anteriores.

2.2 Extração da Placa

A etapa de extração consiste em localizar uma ou mais placas na imagem capturada durante a etapa anterior, e recortá-las para o próximo passo. Dependendo da estrutura do sistema ALPR, pode ser necessário corrigir algumas características da imagem ou aplicar filtros para agilizar o processamento desta e das demais etapas.

Para evitar a necessidade de processar todos os pixels de uma imagem, alguns atributos da placa podem ser usados para auxiliar a sua localização. Atributos como o formato retangular, cor de fundo padronizada, variação entre a cor dos caracteres e do fundo, além da própria existência de letras e números podem ser usados para identificar uma região da imagem que contém uma placa. Em alguns casos pode-se usar dois ou mais atributos combinados para ter maior acurácia na extração. Em (Hongliang e Changping, 2004), os autores demonstram uma versão utilizando somente a localização de bordas verticais, e atingem perto de 100% de sucesso.

Figura 2.2: Detecção de bordas

Na figura 2.2 demonstramos o processo de detecção de bordas aplicado à imagem capturada anteriormente. Na direita foram destacadas as bordas verticais que delimitam a placa. Usando o conhecimento da proporção entre a largura e altura de uma placa válida, é possível determinar que esta é uma região candidata a conter uma placa.

2.3 Segmentação dos Caracteres

A etapa de segmentação dos caracteres consiste em localizar e recortar as letras e números da imagem da placa extraída na etapa anterior. É durante esta etapa que são corrigidos os erros de distorções e rotações apresentados durante a captura da imagem. Também é possível realizar uma primeira avaliação da qualidade da imagem obtida durante esta etapa.

O método mais comum para realizar a segmentação é o de identificação de componentes conexos. Porém, este método apresenta dificuldades no caso de caracteres unidos por resíduos na imagem ou por uma escolha ruim do limiar durante o processo de binarização da imagem. Para diminuir estes problemas a imagem deve ser pré-processada passando por filtros de redução de ruído, normalização de histograma e aumento do contraste.

Durante a construção do dataset usado neste trabalho as placas foram binarizadas manualmente, e os casos de caracteres conexos resolvidos com auxílio de um editor gráfico. Após este pré processamento, um analisador de componentes conexos identificou e separou os caracteres.

Figura 2.3: Segmentação de Caracteres

Na figura 2.3 vemos o resultado da análise de componentes conexos. À esquerda, a placa após o processo de binarização, e à direita os elementos identificados por cores. Após esta etapa os caracteres são ajustados para um tamanho padrão e salvos em arquivos separados.

No caso específico de placas brasileiras, sabemos que uma placa deve conter três letras seguidas de quatro números - no caso de placas de motos, os números aparecem abaixo das letras. Esta padronização possibilita separar os caracteres por tipo antes mesmo da etapa de reconhecimento, evitando problemas de confusão entre o número zero e a letra "O", por exemplo, além de nos permitir avaliar com mais precisão se o processo de segmentação de caracteres foi feito corretamente.

2.4 Reconhecimento dos Caracteres

Finalmente, na etapa de reconhecimento de caracteres, cada imagem é classificada de acordo com a letra ou número que esta representa. Alguns dos

desafios desta etapa incluem a existência de diferentes fontes, problemas e defeitos durante a segmentação dos caracteres, diferença na espessura do traço de acordo com a distância da câmera no momento da captura, entre outros.

Algumas técnicas de processamento de imagem podem ser utilizadas para eliminar ou ao menos diminuir estes problemas. Neste trabalho porém, demonstramos que, com um bom mecanismo de aprendizado de máquinas, estes obstáculos podem ser superados sem necessidade de aplicar mais filtros ou outros processamentos mais complexos.

3
Trabalhos Relacionados

Em (Du et al., 2013), os autores apresentam um relatório com o estado da arte em pesquisas de sistemas ALPR. A tabela 3.1 sintetiza as informações contidas no trabalho dos autores em relação aos estudos sobre reconhecimento de caracteres. A seguir analisamos em maior detalhe dois dos trabalhos citados. Selecionamos o trabalho que apresenta o melhor resultado geral, e o que apresenta o terceiro melhor resultado, uma vez que o segundo melhor aborda o mesmo problema que o primeiro.

(Lee et al., 2004)	Template Matching	95.7%	Taiwan
(Duan et al., 2004)	Hidden Markov Model	97.5%	Vietnam
(Shi et al., 2005)	Neural Network	89.1%	Grécia
(Chang et al., 2004)	-	94.2%	Multi nacional
(Deb e Jo, 2009)	Self Organizing OCR	95.6%	Taiwan
(Wang et al., 2010)	Neural Network	98%	Itália
(Kim et al., 2000)	SVM	97.2%	Coréia
(Comelli et al., 1995)	Template Matching	98.6%	Itália
(Capar e Gokmen, 2006)	Neural Network	97.7%	China

Tabela 3.1: Resumo do estado da arte de reconhecimento de caracteres

3.1
Casamento de Modelos

Em (Comelli et al., 1995), os autores utilizaram casamento de modelos (*template matching*) para reconhecer placas italianas.

O trabalho descrito apresenta uma forte etapa de pré-processamento para obter uma placa cujos caracteres tenham tamanho padronizado e que não apresentem nenhuma inclinação. Após esta etapa é realizado o processo de casamento de modelos para reconhecer os caracteres. Os autores usaram o conhecimento da estrutura da placa italiana, composta de duas letras que representam a província onde o veículo está registrado, e seis caracteres que podem ser todos numéricos ou conter uma (e somente uma) letra. Além disso, todas as placas utilizam a mesma fonte e algumas letras que poderiam causar confusão são excluídas. São elas: C, I, J, O e Q.

A solução foi desenhada de forma que quando houver um certo nível de incerteza na classificação de um caractere, a placa inteira seja rejeitada. Nos caracteres que foram considerados corretos, o sistema apresentou a taxa de acerto de 98.7% para números e 97.97% para letras.

O processo de casamento de modelos requer processamento intensivo, além de um novo modelo base para cada fonte utilizada - aumentando ainda mais o processamento requerido para efetuar o reconhecimento. Além disso, este processo não apresenta um mecanismo de aprendizado continuado, sendo difícil de atualizar o modelo para melhorar a performance do reconhecedor com a sua utilização.

3.2 SVM

Em (Kim et al., 2000), os autores utilizaram *Support Vector Machines* para reconhecer placas Coreanas.

As placas usadas contém duas linhas de caracteres. A primeira apresenta dois caracteres coreanos (que indicam a província de registro do veículo) e dois ou três números. A segunda apresenta um caractere coreano e quatro números.

A solução apresentou taxa de 97.2% e 98% de acerto para os números da primeira e segunda linhas, respectivamente. A taxa de acerto nos caracteres coreanos foi de 98.3% e 95.4%. Não é possível fazer uma comparação direta das tarefas de reconhecimento de caracteres coreanos com o alfabeto latino, portanto nos concentramos apenas no reconhecimento dos números.

Os SVMs têm a mesma complexidade que os Perceptrons para realizar a predição de uma imagem, e portanto têm custo computacional equivalente. Porém, o processo de atualização do Perceptron é muito mais simples, permitindo o aprendizado continuado e, se necessário, de forma *online*. Demonstraremos que, mesmo usando um mecanismo de aprendizado mais simples, nossa taxa de acerto nos números é bem superior ao modelo proposto pelos autores.

4
Solução Proposta

O processo de reconhecimento de caracteres apresenta alguns desafios que devem ser superados para garantir bons níveis de acurácia. Além de defeitos no processo de segmentação de caracteres, alguns problemas como utilização de múltiplas fontes, inclinação da imagem e variação na espessura dos traços tornam a tarefa de reconhecimento mais difícil.

Neste trabalho evitamos utilizar técnicas elaboradas de edição de imagens, focando nossos esforços somente no aspecto de aprendizado de máquinas. Apresentamos aqui as técnicas utilizadas para atingir o objetivo desejado.

4.1
Perceptron Multiclasse

Nossa solução se baseia no aprendizado de máquinas, e mais especificamente no algoritmo do Perceptron multiclasse, um caso particular do Perceptron Estruturado, inicialmente proposto por (Collins, 2002).

Para maior clareza, apresentamos os algoritmos de aprendizado do Perceptron binário (algoritmo 1), do Perceptron Multiclasse (algoritmo 2) e do Averaged Multiclass Perceptron ((algoritmo 3)). No restante deste trabalho nos referimos à estes algoritmos apenas como "algoritmo do Perceptron".

O Perceptron binário é a base dos algoritmos que utilizamos. O nome binário vem do fato de ele ser capaz somente de responder "sim"ou "não"se um exemplo pertence a classe que este reconhece. Para simplificar o algoritmo, ao invés da representação usual de números binários onde 0 significa não e 1 significa sim, alteramos a representação da resposta negativa para -1, de forma que possamos usar a própria resposta diretamente no cálculo de atualização do modelo.

O Perceptron multiclasse é uma extensão do Perceptron proposto por (Rosenblatt, 1958). Considere um exemplo representado por um vetor $X[n]$ de n atributos, e um problema de k classes. Então o Perceptron multiclasse será representado por uma matriz $W[k, n]$, onde cada linha representa um Perceptron, responsável por identificar sua respectiva classe. A função de predição é definida como:

Algoritmo 1 Perceptron Binário

$W \leftarrow 0$ // W é um vetor de tamanho n, onde n é a quantidade de atributos em X
SHUFFLE($DataSet$)

while Não convergir **do**
  for all $(X_i, y_i) \in DataSet$ **do**
    $\hat{y} \leftarrow \text{SIGN}(X_i \times W)$
    if $\hat{y} \neq y_i$ **then**
      $W \leftarrow W + y_i \times X_i$
    end if
  end for
end while

return W

$$\hat{y} = \underset{j}{\text{argmax}}(W \times X)_j$$

E a função de atualização é definida como:

$$W^{t+1} = W^t + \phi(y, X_i) - \phi(\hat{y}, X_i)$$

Onde y denota a classe correta do exemplo, $\hat{y}$ a classe predita e $\phi(x, y)$ é a função que retorna uma matriz $K \times N$ com os valores de y na linha x e as demais linhas zeradas.

Algoritmo 2 Perceptron Multiclasse

$W \leftarrow 0$ // W é uma matriz $[k, n]$, onde n é a quantidade de atributos em X e k a quantidade de classes no $DataSet$
SHUFFLE($DataSet$)

while Não convergir **do**
  for all $(X_i, y_i) \in DataSet$ **do**
    $\hat{y} \leftarrow \underset{j}{\text{argmax}}(W \times X_i)_j$
    if $\hat{y} \neq y_i$ **then**
      $W \leftarrow W + \phi(y, X_i) - \phi(\hat{y}, X_i)$ // $\phi(x, y)$ é a função que retorna uma matriz $K \times N$ com os valores de y na linha x e as demais linhas zeradas
    end if
  end for
end while

return W

Para diminuir os efeitos de *overfitting*, utilizamos a variação Averaged Perceptron, onde a predição é realizada utilizando a média de todos os W

calculados durante a etapa de treinamento. Para não ter que armazenar todos os W vistos, calculamos a média de maneira "Online", isto é, atualizando a média a cada novo registro. Com isso, mantemos somente o último W visto e um W_{avg} que guarda a média atual. Neste caso não podemos realizar a atualização em um passo único, pois devemos manter um contador de atualizações em cada linha de W_{avg}. Portanto, a função de atualização se torna:

$$\begin{aligned} W &\leftarrow W + \phi(y, X_i) - \phi(\hat{y}, X_i) \\ W_{avg}[y]^{(t^y+1)} &\leftarrow \tfrac{t^y}{t^y+1} \times W_{avg}[y]^{t^y} + \tfrac{W[y]}{t^y+1} \\ W_{avg}[\hat{y}]^{(t^{\hat{y}}+1)} &\leftarrow \tfrac{t^{\hat{y}}}{t^{\hat{y}}+1} \times W_{avg}[\hat{y}]^{t^{\hat{y}}} + \tfrac{W[\hat{y}]}{t^{\hat{y}}+1} \end{aligned}$$

Onde $W_{avg}[y]^{t^y}$ representa a linha y da matriz W_{avg} no momento t^y, e t^y é o número de atualizações feitas na linha y.

Algoritmo 3 Online Averaged Multiclass Perceptron

```
1:  W ← 0
2:  W_avg ← 0                                   // W e W_avg são matrizes [K × N]
3:  t ← 0                                       // t é um vetor de tamanho k
4:  SHUFFLE(DataSet)

5:  while Não convergir do
6:      for all (X_i, y_i) ∈ DataSet do
7:          ŷ ← argmax_j (W × X_i)_j
8:          if ŷ ≠ y then
9:              W ← W + φ(y, X_i) − φ(ŷ, X_i)
10:             W_avg[y] ← (t^y/(t^y+1)) × W_avg[y] + W[y]/(t^y+1)
11:             W_avg[ŷ] ← (t^ŷ/(t^ŷ+1)) × W_avg[ŷ] + W[ŷ]/(t^ŷ+1)
12:             t^y ← t^y + 1
13:             t^ŷ ← t^ŷ + 1
14:         end if
15:     end for
16: end while

17: return W_avg
```

Como podemos perceber, tanto a predição quanto a atualização são tarefas de baixa complexidade. De fato, a predição consiste em uma multiplicação de matrizes, e portanto é de ordem $\theta(nk)$. Já o processo de atualização consiste em quatro somas de vetores de tamanho n, portanto é da ordem de $\theta(n)$. Como k é constante - para o reconhecedor de letras $k = 26$ e para o reconhecedor de números $k = 10$ -, o custo da predição fica dependente apenas da quantidade n de atributos utilizados.

Cabe destacar também que, apesar de utilizar o W durante a etapa de treinamento, o Averaged Perceptron usa o W_{avg} para classificar os exemplos não vistos previamente.

Isto demonstra que o Perceptron é um algoritmo apropriado para atingirmos os objetivos estipulados, de excelente performance de predição e possibilidade de atualização online.

4.2 Atributos Básicos

O algoritmo do Perceptron é excelente para resolver problemas linearmente separáveis, porém, ele não é capaz de realizar combinações lineares entre os atributos de entrada. Portanto, é crucial que novos atributos sejam fornecidos para auxiliar o Perceptron durante o processo de classificação.

Estes atributos podem ser gerados manualmente, utilizando o conhecimento do domínio do problema, ou calculados automaticamente, através de combinações dos atributos existentes.

Para atingir o melhor desempenho possível na classificação, é importante selecionar quais atributos serão utilizados. Neste trabalho apresentamos os resultados obtidos usando todos os pixels da imagem, as projeções horizontais e verticais, e o número de pixels pretos em uma janela de tamanho 3×3 proposto em (Aghdasi e Ndungo, 2004).

Na figura 4.1, apresentamos uma visualização dos pixels de um número 5. Um pixel preto é representado por um 1 no dataset gerado, e um pixel branco, por 0. Para a visualização dos demais atributos, utilizaremos seções desta imagem como exemplos.

As projeções consistem e contar o número de pixels ligados - isto é, pixels pretos - em cada linha e coluna. Na figura 4.2 apresentamos as projeções horizontais das primeiras linhas da figura 4.1, e na figura 4.3, as projeções verticais das primeiras colunas. Para melhor visualização, a figura 4.3 foi cortada verticalmente.

Na figura 4.4, apresentamos a visualização dos primeiros blocos da imagem. Observe que existe uma coluna à direita que não está representada na imagem. Esta coluna também é contada, porém, por conter somente três pixels, o valor destes blocos varia de zero a três. No caso desta imagem, todos estes blocos seriam representados por zero.

O processo de geração destes atributos requer um pré-processamento da imagem a ser classificada, o que indiscutivelmente afeta a performance total do algoritmo. Porém, esta etapa adicional possibilita um ganho significativo em termos de acurácia da classificação.

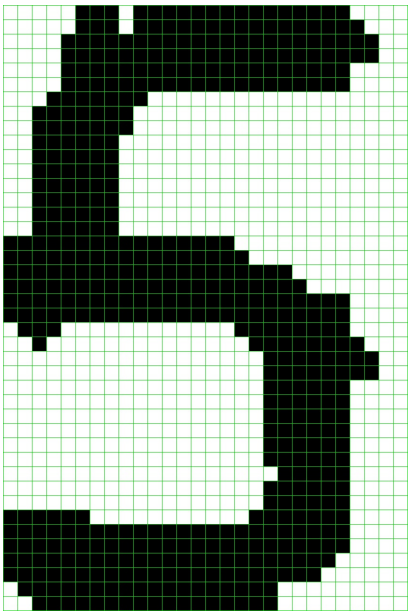

Figura 4.1: Visualização dos pixels de um caractere

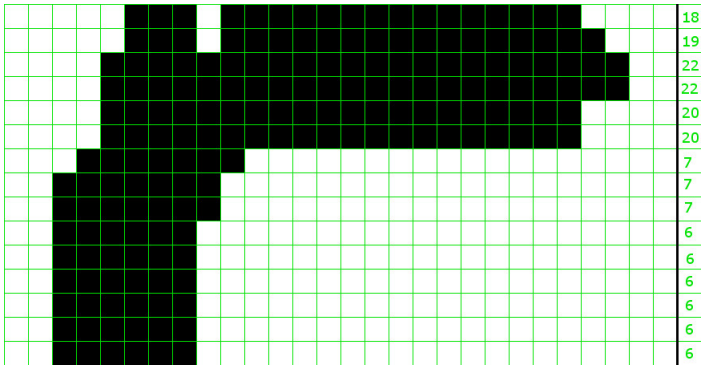

Figura 4.2: Projeções horizontais

A utilização ou não dos atributos gerados pode ser decidida de acordo com a aplicação do algoritmo. Em casos onde a performance é mais importante do que a acurácia, pode-se desejar abrir mão de um conjunto de atributos em prol do menor tempo de processamento.

O algoritmo 4 apresenta a forma de cálculo dos atributos. Cabe ressaltar que os atributos são todos calculados em uma única passagem pela imagem. Após os cálculos dos vetores de atributos, os datasets são construídos usando somente os atributos relevantes.

4.3 Geração de Atributos Guiada por Entropia

A necessidade de conhecimento do domínio do problema para o processo manual de geração de atributos imputa limitações ao Perceptron. Apresentamos outra forma de buscar a separabilidade linear, através da combinação dos

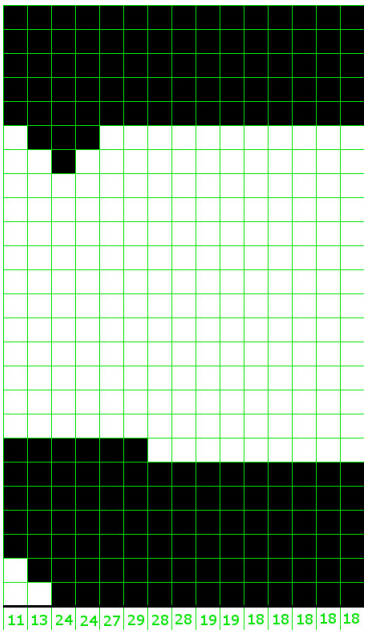

Figura 4.3: Projeções verticais

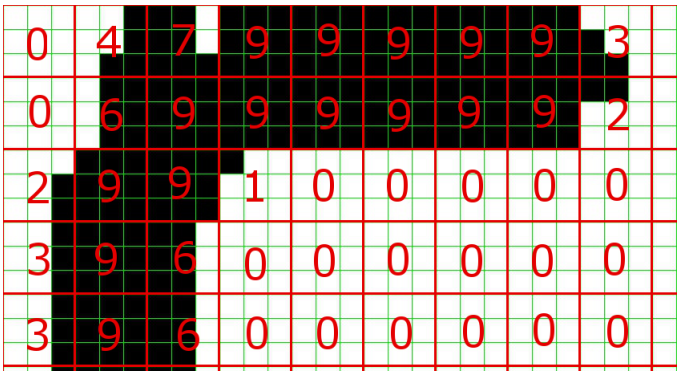

Figura 4.4: Contagem de pixels em blocos de 3×3

atributos já existentes em novos atributos mais complexos.

No nosso trabalho usamos o modelo proposto em (Santos e Milidiú, 2012), que se baseia na entropia dos atributos originais para combiná-los em novos atributos.

Para criar as regras de combinação dos atributos, primeiramente é construída uma árvore de decisão a partir dos dados de treino. Utilizamos o algoritmo C5.0 (See5 - RuleQuest Research) para gerar a árvore. Esta é então utilizada como matriz para a formação das regras de combinação.

A figura 4.5 mostra uma árvore de decisão gerada à partir dos atributos calculados para as imagens. Para a geração das regras de combinação podemos descartar as folhas - que determinam as classes que a árvore atribuiu a cada exemplo - e usar somente as decisões tomadas em cada etapa.

Na tabela 4.1 listamos algumas das regras criadas para a árvore da figura 4.5. Na prática, o algoritmo gera todas as combinações de todos os

Algoritmo 4 Geração Manual de Atributos

1: *Projecoes_Horz* $\leftarrow 0$ // Projeções horizontais - tem o mesmo tamanho que a altura da imagem
2: *Projecoes_Vert* $\leftarrow 0$ // Projeções Verticais - tem o mesmo tamanho que a largura da imagem
3: *Blocos* $\leftarrow 0$ // Cada posição do vetor representa um bloco de 3×3
4: *Pixels* $\leftarrow 0$ // Vetor com os pixels pretos da imagem
5: *blocosPorLinha* $\leftarrow$ TETO($\frac{largura\_imagem}{3}$)

6: **for** l de 0 até *altura_imagem* **do**
7: **for** c de 0 até *largura_imagem* **do**
8: **if** *Imagem_lc* é um pixel preto **then**
9: $Projecoes\_Horz[l]$++
10: $Projecoes\_Vert[c]$++
11: $Blocos[\frac{l}{3} \times blocosPorLinha + \frac{c}{3}]$++
12: $Pixels[l][c] = 1$
13: **end if**
14: **end for**
15: **end for**

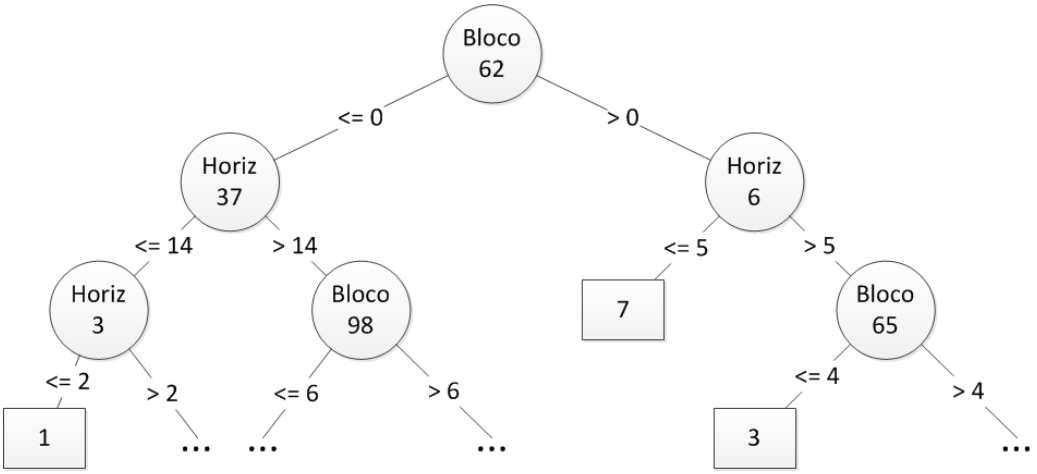

Figura 4.5: Árvore de decisão gerada

caminhos desde a raiz até cada nó, limitados à uma profundidade desejada.

Os novos atributos devem ser gerados para cada exemplo, antes de executar a predição do Perceptron. Desta forma, o tamanho do vetor de entrada passa para $n + g$, onde g é a quantidade de regras criadas no processo descrito acima. Isto também faz com que o modelo passe a ser $W[k, n + g]$.

As demais funções do Perceptron seguem inalteradas, o que significa que mantemos a garantia de convergência dada pelo Perceptron.

O algoritmo 5 apresenta o mecanismo para calcular as regras de geração de atributos. Gerar os atributos é trivial uma vez que se tenha o conjunto de regras necessárias, bastando para isso apenas aplicar todas as regras a cada

Regra	Atributos
1	$Bloco\_62 > 0$
2	$Bloco\_62 <= 0$
3	$Bloco\_62 > 0$ && $Horiz\_37 > 14$
4	$Bloco\_62 <= 0$ && $Horiz\_37 > 14$
5	$Bloco\_62 > 0$ && $Horiz\_37 <= 14$
6	$Bloco\_62 <= 0$ && $Horiz\_37 <= 14$
7	$Bloco\_62 > 0$ && $Horiz\_6 > 5$
8	$Bloco\_62 <= 0$ && $Horiz\_6 > 5$
9	$Bloco\_62 > 0$ && $Horiz\_6 <= 5$
10	$Bloco\_62 <= 0$ && $Horiz\_6 <= 5$

Tabela 4.1: Regras de Geração de Atributos com profundidade 2

Algoritmo 5 Geração de Atributos Guiada por Entropia

```
1: Regras ← {}
2: // 'Raiz' é a raiz da árvore de decisão gerada com base nos exemplos do dataset
3: INSERE(Regras, Raiz, 1, TRUE )

4: procedure INSERE(Regras, No, Profundidade, RegraPai)
5:     if No ≠ NULL && Profundidade < ProfundidadeMaxima then
6:         RegraMenor ← RegraPai && No.valor <= No.Referencia
7:         RegraMaior ← RegraPai && No.valor > No.Referencia

8:         Regras ← add(RegraMenor)
9:         Regras ← add(RegraMaior)

10:        INSERE(Regras, No.Esquerda, Profundidade + 1, RegraMenor)
11:        INSERE(Regras, No.Esquerda, Profundidade + 1, RegraMaior)
12:        INSERE(Regras, No.Direita, Profundidade + 1, RegraMenor)
13:        INSERE(Regras, No.Direita, Profundidade + 1, RegraMaior)
14:    end if
15: end procedure
```

linha do dataset.

4.4 Predição por Comitê

Outra técnica para buscar melhores resultados de classificação, é conhecida como predição por comitê, proposto por (Breiman, 1996).

Com este método são treinados vários Perceptrons simultaneamente, todos com a mesma tarefa de classificação (algoritmo 6). Quando um registro deve ser classificado, os Perceptrons são acionados independentemente e a predição individual é salva. A classe que receber mais votos - isto é, aquela

que for escolhida por mais Perceptrons - é considerada a classe predita pelo comitê (algoritmo 7).

Algoritmo 6 Construção do Comitê

```
1: tam ← 10
2: comite ← pcpts[tam]        // O Comitê nada mais é que um vetor de tam
   perceptrons individuais

3: for all pcpt_i ∈ comite do
4:     pcpt_i ← TRAIN_PERCEPTRON(DataSet)
5: end for

6: return comite
```

Com o intuito de reduzir o número de execuções desnecessárias, os Perceptrons são acionados sequencialmente, e quando já não é mais possível reverter a votação, o processo é interrompido. Desta forma, um comitê de k Perceptrons só executa k classificações em casos muito equilibrados. Nos casos mais fáceis, bastam $\frac{k}{2}$ classificações para se chegar a um veredito.

Algoritmo 7 Predição do Comitê

```
1: tam ← 10
2: prim ← 0   // Guarda o número de votos da classe com mais votos até o momento
3: seg ← 0  // Guarda o número de votos da segunda classe com mais votos ate o momento
4: classePrim ← 0          // Guarda a classe que está com mais votos até o momento
5: classeSeg ← 0          // Guarda a classe que está em segundo lugar até o momento
6: votos ← 0     // Vetor de k posições, guarda o número de votos que cada classe recebeu até o momento

7: for i de 0 até (tam − 1) do
8:     if prim >= seg + (tam − i) then
9:         break
10:    end if

11:    ŷ ← pcpt_i.predict(X)          // pcpt_i é o iésimo perceptron no comitê
12:    votos[ŷ]++

13:    if votos[ŷ] > prim then
14:        if ŷ ≠ classePrim then
15:            seg ← prim
16:            classeSeg ← classePrim
17:        end if
18:        prim ← votos[ŷ]
19:        classePrim ← ŷ
20:    end if
21: end for

22: return classePrim
```

5
Descrição do Dataset

Para este trabalho usamos um dataset composto de 600 (seiscentas) imagens de câmeras fotográficas, tanto de avanço de sinal quanto de limite de velocidade. As imagens contém fotos de carros de passeio, táxis, motocicletas, ônibus e caminhões. Apesar da diferença de organização das placas de motocicletas, o pré-processamento realizado consegue separar as letras e os números da mesma forma.

Destas 600 imagens pudemos retirar 2.400 exemplos de números e 1.200 exemplos de letras. Apesar de uma placa brasileira conter três letras e quatro números, descartamos a primeira letra de cada placa, pois existe muita repetição uma vez que a maioria das placas de um estado começam pela mesma letra (estados maiores como Rio de Janeiro e São Paulo possuem duas possibilidades de letras iniciais, mas ainda causariam um desbalanceamento no dataset).

5.1
Pré-Processamento

Antes de podermos executar o processo de reconhecimento dos caracteres, foi necessário pré-processar as imagens obtidas. Na figura 5.1 vemos um exemplo de uma imagem de avanço de sinal, antes de ser processada para as etapas seguintes.

Com o auxílio do editor gráfico *Gimp* (GIMP – The GNU Image Manipulation Program), os seguintes passos foram executados:

1. Recortar a área da placa

   Usando a ferramenta *Crop*, extraímos somente a região da placa, como demonstrado na figura 5.2.

2. Ajustar a rotação da placa

   Dependendo do ângulo entre a câmera e o veículo, a placa pode aparecer inclinada. Corrigimos este problema utilizando a ferramenta *Rotate*, como demonstra a figura 5.3

Figura 5.1: Exemplo de imagem de câmera de avanço de sinal

Figura 5.2: Ferramenta Crop

3. Binarizar a imagem

   Utilizamos a função *Threshold* para binarizar a imagem. Esta ferramenta faz com que todos os pixels com intensidade abaixo de um valor (limiar) sejam convertido para um pixel totalmente preto, e os acima do limiar são convertidos para um pixel totalmente branco. Não foi usado nenhum processo automático para ajustar o valor ideal do limiar, que foi escolhido manualmente caso a caso. A ferramenta apresenta controle gráfico que auxilia na hora de escolher o valor mais apropriado.

4. (Opcional) Inverter as cores de texto e fundo

   Nos casos de veículos comerciais e de transporte de passageiros, é necessário fazer a inversão das cores.Isso porque nestes casos a placa tem fundo vermelho e letras brancas, o que faz com que a imagem após o

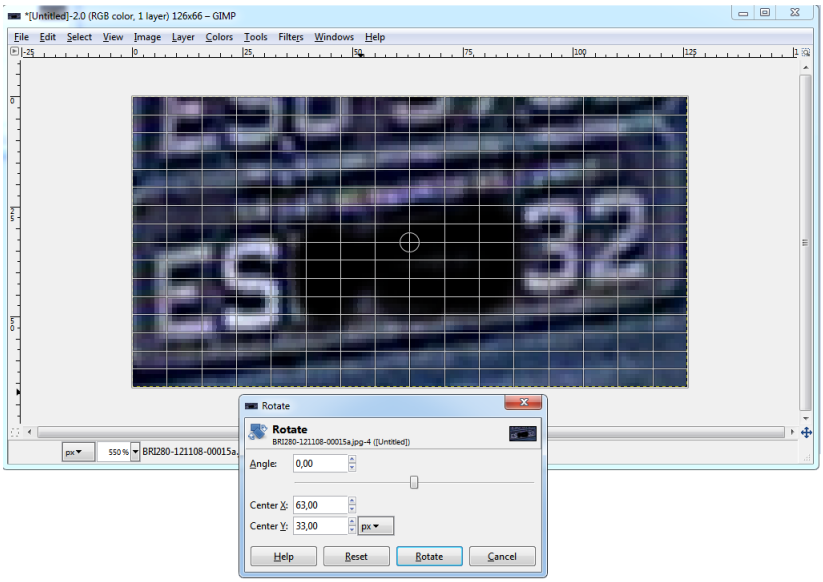

Figura 5.3: Ferramenta Rotate

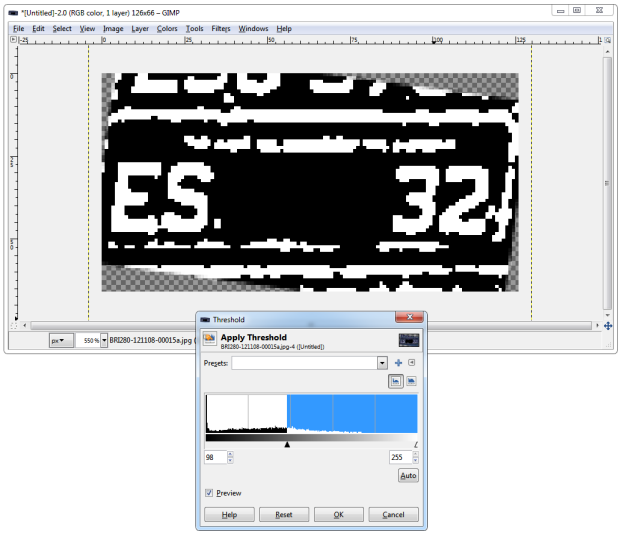

Figura 5.4: Ferramenta Threshold

processo de limiar fique invertida em relação ao padrão (i.e. fundo preto e letras brancas). Utilizamos a função *Invert*, conforme demonstrado na figura 5.5.

5. (Opcional) Separação de caracteres conectados

   Por fim, é necessário separar os casos onde dois caracteres aparecem unidos por algum ruído na imagem original. A figura 5.6 mostra um exemplo deste tipo de problema. Note que o 'M' e o 'W' aparecem conectados pelo topo das letras. Para realizar a separação, basta traçar um risco branco onde os caracteres aparecem conectados.

Nesta etapa já foi feita uma filtragem removendo as imagens de qualidade

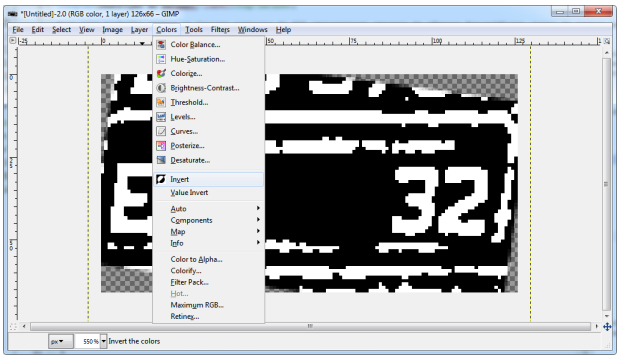

Figura 5.5: Ferramenta Invert

Figura 5.6: Exemplo de imagem com caracteres unidos por ruído

muito baixa, que tivessem os caracteres ilegíveis antes do pré-processamento, ou após a placa ser binarizada.

5.2 Segmentação de Caracteres

Uma vez que temos uma imagem da placa binarizada e sem rotações muito grandes, podemos iniciar o processo de segmentação de caracteres.

Apesar de não ser escopo principal deste trabalho, implementamos um processo automático para esta etapa, a fim de diminuir a necessidade de trabalho manual repetitivo.

Nossa implementação consiste em separar os componentes conexos da imagem. Para isto realizamos uma busca na imagem por pixels pretos, trocando a cor do pixel encontrado e todos os seus vizinhos para uma nova cor. O algoritmo 8 demonstra o processo em maior detalhe.

Ao término da execução, cada bloco conexo terá uma cor diferente. Nossa implementação usa características simples para tentar identificar quais blocos representam caracteres. Os blocos que se enquadrarem em um padrão de tamanho, posição e proporção é considerado um caractere. Caso não sejam identificados sete caracteres, a imagem é considerada inválida e marcada para revisão manual.

Nos casos em que a imagem foi considerada válida, cada um dos blocos é salvo em um arquivo individual separado, para gerar os datasets de letras e números. Pela posição de cada bloco é possível determinar se a imagem

Algoritmo 8 Identificação de componentes conexos

```
1: cor ← 1

2: for (y ← 0; y < largura; y ← y + 1) do
3:     for (x ← 0; x < largura; x ← x + 1) do
4:         if pixel[x, y] é um pixel preto then
5:             MUDA_COR(x, y, cor)
6:             cor ← cor + 1
7:         end if
8:     end for
9: end for

10: procedure MUDA_COR (x, y, cor)
11:     pixel[x, y] ← cor
12:     if pixel[x + 1, y] é um pixel preto then
13:         MUDA_COR(x + 1, y, cor)
14:     end if
15:     if pixel[x − 1, y] é um pixel preto then
16:         MUDA_COR(x − 1, y, cor)
17:     end if
18:     if pixel[x, y + 1] é um pixel preto then
19:         MUDA_COR(x, y + 1, cor)
20:     end if
21:     if pixel[x, y − 1] é um pixel preto then
22:         MUDA_COR(x, y − 1, cor)
23:     end if
24: end procedure
```

representa uma letra ou número. Usando a distância do bloco até o topo da imagem como primeiro critério de ordenação e a distância para a margem esquerda como segundo critério, pudemos utilizar o mesmo método para processar placas de carros e motos. Com exceção das placas de motocicletas, todos os caracteres têm a mesma distância do topo, e portanto o três mais próximos à margem esquerda são letras e os quatro demais são números. No caso das motos, os três mais próximos da margem superior são letras, e os demais, números.

Cada caractere identificado foi redimensionado utilizando o método *cvResize* da biblioteca OpenCV (Bradski, 2000), para o tamanho padrão de 28×42 pixels. A proporção entre largura e altura das letras e números foi mantida durante o redimensionamento. Os blocos foram centralizados no novo tamanho para manter uma padronização em relação ao centro da imagem.

5.3 Anotação de Exemplos

Com cada caractere separado em imagens individuais, precisamos apenas anotar o conteúdo de cada imagem para podermos usá-las no experimento.

Para poder anotar todos as 3.600 imagens mais facilmente, montamos um sistema de *crowd sourcing*, onde disponibilizamos um site no qual qualquer pessoa pode indicar a letra ou número representado em cada imagem (Figura 5.7).

Figura 5.7: Tela do site de anotação de exemplos

Para evitar problemas de falha na digitação, o sistema solicita a anotação de uma mesma imagem duas vezes, e somente quando as duas respostas são iguais o sistema considera a imagem anotada. A cada acesso é sorteada uma nova imagem, para diminuir as chances de um mesmo usuário dar as duas anotações para um exemplo.

É importante ressaltar que apesar de todas as placas usadas serem brasileiras, existem diversas fontes em uso. A figura 5.8 mostra alguns exemplos de diferenças entre os números de diversas fontes. Não foi feito nenhum tratamento para diferenciar uma fonte da outra durante a anotação. O algoritmo de classificação deve ser robusto o bastante para tratar mais de uma fonte para o mesmo caractere.

Durante o processo de anotação o usuário pode identificar um caractere ilegível ou incompleto por falha do processo de segmentação. Nestes casos

Figura 5.8: Exemplos de fontes diferentes em uso no sistema brasileiro

ele pode escolher a opção "Editar"para que a imagem seja reprocessada manualmente ou excluída da base.

6
Avaliação Empírica

Neste capítulo apresentamos os experimentos realizados para avaliar a eficácia da solução proposta. Começamos pela discriminação dos conjuntos de testes elaborados, em seguida apresentamos os resultados obtidos e por fim fazemos uma análise dos erros de classificação cometidos pelo melhor classificador construído.

Analisando os resultados obtidos fazemos uma avaliação dos conjuntos de atributos que se demonstraram mais eficazes para o aprendizado.

Através da análise de erros apresentamos alguns métodos para melhorar a acurácia do preditor proposto.

6.1
Experimentos

Para o experimento, separamos a tarefa de reconhecimento de letras da tarefa de reconhecimento de números. Desta forma diminuímos confusões entre alguns caracteres que sejam muito similares, como a letra "I"e o número "1"ou a letra "O"e o número 0, que de fato, são idênticos na fonte utilizada em algumas placas brasileiras.

Para realizar os testes, construímos quatro datasets distintos para cada tarefa, em um total de oito datasets. Abaixo uma breve descrição do conteúdo de cada um:

Pixels O dataset mais simples. Contém somente os pixels de cada imagem, após o pré-processamento descrito no capítulo 5.

Pixels + Projeções Além dos pixels, incluímos 70 novos atributos referentes à contagem de pixels pretos em cada linha e em cada coluna.

Pixels + Projeções + Blocos Inclui todos os atributos anteriores e mais 140 referentes à contagem de pixels pretos em blocos de 3×3. O último bloco de cada linha fica incompleto, porém isso não é um problema para a contagem.

Projeções + Blocos Com o objetivo de diminuir a quantidade de atributos, retiramos os pixels originais das imagens, mantendo apenas os atributos construídos nos itens anteriores.

Para cada um dos datasets descritos anteriormente, executamos duas versões. Primeiro utilizamos somente os atributos contidos no próprio dataset. Na segunda execução, utilizamos o método de geração automática de atributos (EFG) para tentar melhorar os resultados obtidos.

Em seguida repetimos os testes utilizando um comitê composto por dez Perceptrons. O processo de medição da acurácia é o mesmo, sendo alterado somente o processo de classificação.

6.2 Resultados

Para medir a acurácia obtida em cada experimento, utilizamos o método de validação cruzada *leave one out*. Neste método, o modelo é treinado em $m - 1$ exemplos, onde m é o total de exemplos disponíveis no dataset. Em seguida, o modelo aprendido é testado contra o exemplo excluído do treino. O processo é repetido para todos os elementos do dataset, e a estimativa de acurácia da predição do modelo para elementos não vistos durante o treino é medida pela fórmula:

$$Acc = 1 - \frac{1}{m} \sum_{i=1}^{m} Err(D_i, Dt_i)$$

Onde m é o número total de exemplos, D_i é o dataset composto por todos os exemplos exceto o exemplo i, Dt_i é o dataset composto somente pelo exemplo i e $Err(x, y)$ é a função que retorna o número de erros no dataset y quando o modelo é treinado utilizando o dataset x.

O algoritmo 9 demonstra o método de validação cruzada para testar um Perceptron. Para o teste do comitê, basta alterar a chamada da função de treino e executar a predição do comitê ao invés da predição do Perceptron. Os demais procedimentos permanecem inalterados.

Como ainda assim estamos lidando com datasets pequenos, o resultado final sofre interferência em função da ordem que os elementos são apresentados aos Perceptrons durante a etapa de treino. Para amortizar esta influência, executamos cada teste dez vezes e reportamos apenas a média aritmética das acurácias obtidas em cada execução.

Para calcular a acurácia total do experimento em cada versão, fizemos a média ponderada da acurácia da classificação de números e de letras. Como em nosso experimentos tínhamos o dobro de números, o total é calculado como:

Algoritmo 9 Validação Cruzada *leave one out*

```
1: Qtd_Erros ← 0

2: for all (X_i, y_i) ∈ DataSet do
3:     DataSet_Treino ← DataSet − X_i
4:     DataSet_Teste ← X_i

5:     W ← Run_Perceptron(DataSet_Treino)
6:     ŷ = argmax_i (W × X)_i
7:
8:     if ŷ ≠ X_i.classe then
9:         Qtd_Erros ← Qtd_Erros + 1
10:    end if
11: end for

12: Acuracia ← (1 − Qtd_Erros / DataSet.Size) × 100
13: return Acuracia
```

$$acc\_total \leftarrow \frac{2 \times acc\_numeros + acc\_letras}{3}$$

Onde *acc_total* é a acurácia total estimada para qualquer caractere, *acc_numeros* é a acurácia medida no preditor de números e *acc_letras* é a acurácia medida no preditor de letras.

Incluímos nas tabelas 6.1 e 6.2 os resultados dos trabalhos relacionados vistos no capítulo 3, ordenados em comparação com os nossos resultados obtidos em nosso experimento. Para maior clareza em relação aos efeitos dos atributos gerados e do EFG, nossos resultados foram ordenados pelo dataset utilizado, e não pelo resultado obtido. Isto não atrapalha a comparação com os resultados dos outros trabalhos.

Na tabela 6.1 apresentamos os resultados obtidos em cada um dos dezesseis casos de teste realizados com apenas um Perceptron. Nesta tabela já podemos perceber a eficiência do método utilizado, já que a estratégia mais simples já parte em igualdade com um dos trabalhos citados.

A tabela 6.2 mostra os resultados obtidos em cada um dos dezesseis casos de teste realizados com um comitê composto de dez Perceptrons. Observe que o pior resultado com o comitê já é superior ao melhor resultado obtido com somente um Perceptron. Estes resultados evidenciam também que os novos atributos inseridos são dominados pelos pixels da imagem, porém temos um ganho significativo quando usamos somente os atributos calculados.

Na tabela 6.2 incluímos uma versão do nosso melhor modelo que desconsidera as letras inexistentes nas placas italianas. Isto nos permite fazer uma comparação aproximada com o trabalho de (Comelli et al., 1995), porém, por

Dataset	Acc. Números	Acc. Letras	Total
(Kim et al., 2000)	**-**	**-**	**97,2%**
Pixel	98,7%	94,2%	97,2%
Pixel + EFG	98,7%	94,7%	97,4%
Pixel_Proj	98,7%	94,3%	97,2%
Pixel_Proj + EFG	98,7%	94,4%	97,3%
Pixel_Proj_Bloc	98,7%	94,5%	97,3%
Pixel_Proj_Bloc + EFG	98,8%	94,6%	97,4%
(Duan et al., 2004)	**-**	**-**	**97,5%**
(Capar e Gokmen, 2006)	**-**	**-**	**97,7%**
Proj_Bloc	98,9%	95,5%	97,8%
Proj_Bloc + EFG	99,0%	95,6%	97,8%

Tabela 6.1: Resultados dos experimentos realizados com um Perceptron

Dataset	Acc. Números	Acc. Letras	Total
(Wang et al., 2010)	**-**	**-**	**98%**
Pixel	99,1%	96,0%	98,1%
Pixel + EFG	99,1%	96,0%	98,1%
Pixel_Proj	99,1%	96,1%	98,1%
Pixel_Proj + EFG	99,1%	96,1%	98,1%
Pixel_Proj_Bloc	99,0%	96,0%	98,0%
Pixel_Proj_Bloc + EFG	99,1%	96,1%	98,1%
Proj_Bloc	99,1%	96,5%	98,2%
Proj_Bloc + EFG	99,2%	96,6%	98,3%
(Comelli et al., 1995)	**-**	**-**	**98,6%**
Proj_Bloc + EFG (Itália)	99,2%	97,4%	98,6%

Tabela 6.2: Resultados dos experimentos realizados com o comitê

diferenças na resolução das imagens e condições de captura, ainda não devemos considerar os dois problemas como idênticos.

6.3 Análise de Erros

Analisando os erros cometidos pelos classificadores propostos, podemos avaliar quais medidas podem ser tomadas no processo de treinamento para obter melhores resultados na predição.

Para nos auxiliar neste estudo, apresentamos a matriz de confusão da classificação dos números (tabela 6.3) e das letras (tabela 6.5), além das tabelas 6.4 e 6.6 que totalizam o resultado de um ciclo de teste utilizando o método do comitê e o dataset que contém as projeções e contagem de pixels por blocos, além dos atributos gerados pelo EFG. Lembramos que os totais representados nestas tabelas não necessariamente condizem com o total reportado no experimento pois estes foram extraídos de apenas uma execução,

enquanto o total reportado é calculado pela média de várias execuções.

		Classificada como										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Acurácia**
Número da Imagem	**0**	257	0	0	0	0	0	0	0	0	0	100,0%
	1	0	245	0	0	0	0	0	1	0	0	99,6%
	2	0	0	242	0	0	0	0	0	0	0	100,0%
	3	0	0	0	211	0	0	0	0	0	1	99,5%
	4	0	0	1	0	225	0	1	0	0	1	98,7%
	5	1	0	0	0	0	230	3	0	0	0	98,3%
	6	0	0	0	0	0	2	240	0	1	0	98,8%
	7	0	0	0	1	0	0	0	244	0	0	99,6%
	8	0	0	0	0	0	0	1	0	231	0	99,6%
	9	0	0	0	1	0	0	0	0	2	258	98,9%

Tabela 6.3: Matriz de confusão dos números

Número	Total	Corretos	Errados	Acurácia
0	259	259	0	100,0%
1	246	245	1	99,6%
2	244	244	0	100,0%
3	211	210	1	99,5%
4	226	224	3	98,7%
5	233	230	4	98,3%
6	244	241	3	98,8%
7	245	244	1	99,6%
8	232	228	1	99,6%
9	260	258	3	98,9%
Total	2400	2383	17	99,3%

Tabela 6.4: Resultados de classificação dos números em um experimento realizado com o comitê

Observando a matriz de confusão gerada na classificação dos números, vemos que os algarismos 5 e 6 se confundem.

Na figura 6.1 (a), apresentamos a imagem de um número 5 que foi classificado como 6. A linha mais espessa da imagem e a proximidade entre o a linha central e inferior do número, levaram o Perceptron a confundir com o algarismo 6 com uma leve falha na parte inferior à esquerda. Avaliando as imagens de entrada, percebemos que existem poucos exemplos de números 5 que apresentam estas características. Para decidir a ação a ser tomada, temos que considerar a ocorrência destas características em imagens reais. Caso não seja frequente, podemos considerar um ruído aceitável, mas caso seja mais comum, podemos tentar apresentar mais exemplos semelhantes à este durante

o treino, ou efetuar um pré-processamento de erosão na imagem de entrada para diminuir a espessura dos traços.

(a) (b)

Figura 6.1: Números categorizados erradamente

Já na figura 6.1 (b) ocorre o inverso: o número 6 é reconhecido como 5. Se olharmos com atenção, podemos ver que a imagem apresenta falha na mesma região que a figura (a), e poderia ser confundida com um número 5 com ruído. Neste caso, o problema foi causado por uma imagem original de baixa resolução, causando problemas na segmentação de caracteres que foram acentuados pelo processo de binarização. Novamente a correção viria de pré-processamentos mais rebuscados, neste caso para completar os pedaços perdidos.

Com relação à classificação de letras, as tabelas 6.5 e 6.6 nos dão algumas informações importantes. Primeiro, é notório que o dataset não é balanceado. Apesar de termos usado amostras aleatórias durante a construção do dataset, percebemos que algumas letras aparecem com mais frequência do que outras, como o "E"que aparece duas vezes mais que o "Q"ou "S". Outro dado relevante que podemos observar é a baixa acurácia nas letras "D", "O"e "Q". Por fim destacamos "W", "V"e "Y"como letras que podem ser confundidas entre si, principalmente em função de imagens com baixa resolução.

		Classificada Como															
		B	**D**	**F**	**H**	**J**	**K**	**M**	**N**	**O**	**P**	**Q**	**R**	**V**	**W**	**Y**	**Acurácia**
Letra na Imagem	**B**	34	2	0	0	0	0	0	0	0	0	0	0	1	1	0	89,5%
	D	0	28	0	0	0	0	0	0	4	0	2	0	0	0	0	82,4%
	F	0	0	43	0	0	0	0	0	0	1	0	0	1	0	0	95,6%
	N	1	0	0	1	0	0	1	52	0	0	0	0	0	0	0	94,5%
	O	0	2	0	0	0	0	0	0	49	0	1	0	0	0	0	94,2%
	Q	0	1	0	0	0	0	0	0	2	0	23	0	0	0	0	88,5%
	R	0	0	0	0	1	0	0	0	0	0	0	37	0	1	0	94,9%
	V	0	0	0	0	0	1	0	0	0	0	0	0	40	1	2	90,9%
	W	0	0	0	0	0	0	0	1	0	0	0	0	3	27	0	87,1%

Tabela 6.5: Matriz de confusão das letras. Nota: A matriz foi simplificada para melhor visualização, permanecendo somente as linhas que apresentam pior acurácia e colunas que não tivessem todas as posições zeradas.

Letra	Total	Corretos	Errados	Acurácia
A	53	52	1	98,1%
B	38	34	4	89,5%
C	49	49	0	100,0%
D	34	28	6	82,4%
E	58	58	0	100,0%
F	45	43	2	95,6%
G	57	56	1	98,2%
H	50	48	2	96,0%
I	56	56	0	100,0%
J	56	56	0	100,0%
K	53	53	0	100,0%
L	55	55	0	100,0%
M	55	53	2	96,4%
N	55	52	3	94,5%
O	52	49	3	94,2%
P	41	41	0	100,0%
Q	26	23	3	88,5%
R	39	37	2	94,9%
S	31	30	1	96,8%
T	44	44	0	100,0%
U	48	48	0	100,0%
V	44	40	4	90,9%
W	31	27	4	87,1%
X	37	36	1	97,3%
Y	44	44	0	100,0%
Z	49	49	0	100,0%
Total	1200	1161	39	96,8%

Tabela 6.6: Resultados de classificação das letras em um experimento realizado com o comitê

A figura 6.2 apresenta alguns erros que aconteceram na identificação de letras. As imagens representam as letras 'D', 'Y' e 'Q', e foram reconhecidas como 'O', 'T' e 'O', respectivamente.

(a) (b) (c)

Figura 6.2: Letras categorizadas erradamente

No caso das letras o problema é mais complexo de ser resolvido. Claramente o sistema se beneficiaria de mais exemplos para treino, uma vez que o dataset contém somente 34 exemplos de 'D' e 26 exemplos de 'Q'. Além disso,

a fonte utilizada na maioria das placas não apresenta diferenças significativas entre o 'D', 'Q' e 'O', principalmente após o processo de binarização. Note que o 'D' se assemelha muito ao 'O' quando apresenta alguma inclinação. E os exemplos de 'Q' se diferenciam apenas por um leve espessamento da linha na parte inferior direita. A imagem do 'Y' apresenta grande deformação em relação aos outros exemplos, também causada pela baixa resolução da imagem no momento da captura.

Para tentar solucionar os casos do 'Y' e dos 'Q's, pode-se usar o processo de esqueletização da imagem, para chegar à um traço mais fino e que dê maior destaque para a estrutura das letras. Para o caso do 'D', uma detecção e resolução de caracteres inclinados se apresenta mais promissora.

7
Conclusões

Os experimentos realizados com o comitê mostram que a técnica apresenta ganho significativo, ao custo de aumentar o tempo de processamento de cada caractere.

É importante ressaltar que a comparação direta com os trabalhos relacionados deve ser feita com cautela, uma vez que os problemas tratam de datasets distintos, apresentando diferentes desafios devido ao contexto e pré processamentos realizados. A maioria dos trabalhos se baseia em um forte pré-processamento da imagem durante a etapa de segmentação, resultando em caracteres mais "limpos"para a etapa de classificação. Além disso, o próprio contexto das placas tornam as tarefas diferentes. Alguns países evitam usar letras que causem confusão entre si ou com números, o que facilita a tarefa de reconhecimento. Em outros casos não é possível saber de antemão se o caractere é um número ou letra, adicionando complexidade ao classificador utilizado.

Quanto à performance dos atributos gerados manualmente e automaticamente, vamos analisar os resultados separadamente para o reconhecedor de números e o de letras.

7.1
Reconhecedor de Números

Nos testes realizados com os dois primeiros datasets - isto é, o dataset que contém somente os pixels da imagem e o que contém pixels e projeções - vemos que o EFG não apresenta melhoria no resultado da classificação. Isto se dá porque a combinação da informação de alguns pixels e das projeções não gera conhecimento novo aproveitável a respeito do problema.

Nos dois experimentos seguintes, agregando o atributo gerado pela contagem de pixels em blocos, vemos que a simples adição do atributo não provocou uma variação significativa no resultado. Porém, ao acrescentarmos os atributos gerados automaticamente, já podemos perceber alterações no resultado obtido. Usando os pixels e os atributos calculados manualmente, o EFG reduziu a taxa de erro em cerca de 7% para o Perceptron e 10% para o comitê.

Por fim, ao retirarmos os pixels do conjunto de atributos e usarmos somente os atributos gerados manualmente, o comitê obtém mais salto no resultado. Isso se dá pois o modelo gerado sem os pixels sofre menos com *overfitting*, criando um modelo que generaliza melhor o conhecimento adquirido durante a etapa de treinamento. Este também é o experimento em que o EFG apresentou o maior ganho, reduzindo a taxa de erro em mais de 11%.

Se compararmos o melhor modelo gerado com o mais básico, a variação total da taxa de erro ficou acima de 38%.

7.2 Reconhecedor de Letras

O desempenho inferior em relação ao Reconhecedor de Números era previsto. Além da quantidade menor de exemplos disponíveis para o treinamento, a tarefa de reconhecer letras é inerentemente mais complexa, uma vez que o dataset contém mais classes, e algumas letras são facilmente confundíveis entre si, como os grupos {"O", "Q", "D"} e {"W", "V"}.

Ainda assim, é importante observar que o padrão de resultados se mantém. O melhor resultado do preditor de letras também vem do comitê, utilizando somente os atributos gerados manualmente (i.e. sem os pixels) e aplicando o EFG para a geração automática dos demais atributos. Os ganhos do EFG são proporcionalmente menores, porém não desprezíveis. Na comparação entre o melhor e pior modelos, a variação da taxa de erro é de 41%.

Futuros esforços para melhorar a acurácia total do modelo proposto devem se concentrar mais no reconhecedor de letras, uma vez que pequenas melhoras nesta sub-tarefa apresentaram ganhos maiores no resultado final.

7.3 Conclusão Final

Demonstramos que o Perceptron atende a todos os objetivos estabelecidos em relação à performance, acurácia e facilidade de aprendizado contínuo. Observamos porém que o uso de um comitê oferece vantagem significativa no quesito da acurácia, em troca de redução na performance.

Comprovamos também que o mecanismo do EFG pode ser aplicado à tarefas de visão computacional com sucesso.

7.4 Trabalhos Futuros

Novas pesquisas com o objetivo de melhorar os resultados obtidos neste trabalho podem incluir algoritmos de pré processamento das imagens mais elaborados, como a correção da inclinação dos caracteres (Deb et al., 2010), melhoria (*enhancement*) da imagem (Zhang e Zhang, 2003) e esqueletização dos caracteres (You e Tang, 2007).

Também é possível explorar o reconhecimento através da análise de imagens em tons de cinza, ao invés de imagens binarizadas, para diminuir a perda de resolução durante o pré processamento.

Outra área que pode ser explorada é a geração automática de novos exemplos, para aumentar artificialmente o dataset, como proposto em (Bastien et al., 2010).

Por fim, novos métodos de geração de atributos podem ser testados, sejam eles calculados manualmente - como a avaliação do ângulo das retas em sub blocos da imagem, proposto em (Nukano et al., 2004) - ou automaticamente, como os estudos baseados em *deep learning*.

8
Bibliografia

AGHDASI, F.; NDUNGO, H. Automatic licence plate recognition system. In: **AFRICON, 2004. 7th AFRICON Conference in Africa**. [S.l.: s.n.], 2004. v. 1, p. 45–50 Vol.1.

Bastien, F. et al. Deep Self-Taught Learning for Handwritten Character Recognition. **ArXiv e-prints**, set. 2010.

BRADSKI, G. Opencv. **Dr. Dobb's Journal of Software Tools**, 2000.

BREIMAN, L. Bagging predictors. **Machine Learning**, Kluwer Academic Publishers-Plenum Publishers, v. 24, n. 2, p. 123–140, 1996. ISSN 0885-6125.

CAPAR, A.; GOKMEN, M. Concurrent segmentation and recognition with shape-driven fast marching methods. In: **Pattern Recognition, 2006. ICPR 2006. 18th International Conference on**. [S.l.: s.n.], 2006. v. 1, p. 155–158. ISSN 1051-4651.

CHANG, S.-L. et al. Automatic license plate recognition. **Intelligent Transportation Systems, IEEE Transactions on**, v. 5, n. 1, p. 42–53, March 2004. ISSN 1524-9050.

COLLINS, M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In: **Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10**. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002. (EMNLP '02), p. 1–8. Disponível em: <http://dx.doi.org/10.3115/1118693.1118694>.

COMELLI, P. et al. Optical recognition of motor vehicle license plates. **Vehicular Technology, IEEE Transactions on**, v. 44, n. 4, p. 790–799, Nov 1995. ISSN 0018-9545.

DEB, K.; JO, K.-H. A vehicle license plate detection method for intelligent transportation system applications. **Cybernetics and Systems**, v. 40, n. 8, p. 689–705, 2009.

DEB, K. et al. Projection and least square fitting with perpendicular offsets based vehicle license plate tilt correction. In: **SICE Annual Conference 2010, Proceedings of**. [S.l.: s.n.], 2010. p. 3291–3298.

DU, S. et al. Automatic license plate recognition (alpr): A state-of-the-art review. **Circuits and Systems for Video Technology, IEEE Transactions on**, v. 23, n. 2, p. 311–325, Feb 2013. ISSN 1051-8215.

DUAN, T. D.; DUC, D. A.; DU, T. L. H. Combining hough transform and contour algorithm for detecting vehicles' license-plates. In: **Intelligent Multimedia, Video and Speech Processing, 2004. Proceedings of 2004 International Symposium on**. [S.l.: s.n.], 2004. p. 747–750.

GIMP – The GNU Image Manipulation Program. `http://www.gimp.org/`.

HONGLIANG, B.; CHANGPING, L. A hybrid license plate extraction method based on edge statistics and morphology. In: **Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on**. [S.l.: s.n.], 2004. v. 2, p. 831–834 Vol.2. ISSN 1051-4651.

KIM, K. K. et al. Learning-based approach for license plate recognition. In: **Neural Networks for Signal Processing X, 2000. Proceedings of the 2000 IEEE Signal Processing Society Workshop**. [S.l.: s.n.], 2000. v. 2, p. 614–623 vol.2. ISSN 1089-3555.

LEE, H.-J.; CHEN, S.-Y.; WANG, S.-Z. Extraction and recognition of license plates of motorcycles and vehicles on highways. In: **Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on**. [S.l.: s.n.], 2004. v. 4, p. 356–359 Vol.4. ISSN 1051-4651.

NUKANO, T.; FUKUMI, M.; KHALID, M. Vehicle license plate character recognition by neural networks. In: **Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004. Proceedings of 2004 International Symposium on**. [S.l.: s.n.], 2004. p. 771–775.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6, p. 386–408, 1958.

SANTOS, C. N. dos; MILIDIú, R. L. **Entropy Guided Transformation Learning - Algorithms and Applications.** [S.l.]: Springer, 2012. I-XIII, 1-78 p. (Springer Briefs in Computer Science). ISBN 978-1-4471-2977-6.

SEE5 - RuleQuest Research. `http://www.rulequest.com/see5-info.html`.

SHI, X.; ZHAO, W.; SHEN, Y. Automatic license plate recognition system based on color image processing. In: **Proceedings of the 2005 International Conference on Computational Science and Its Applications - Volume Part IV**. Berlin, Heidelberg: Springer-Verlag, 2005. (ICCSA'05), p. 1159–1168. ISBN 3-540-25863-9, 978-3-540-25863-6. Disponível em: <http://dx.doi.org/10.1007/11424925_121>.

WANG, M.-L. et al. A vehicle license plate recognition system based on spatial/frequency domain filtering and neural networks. In: PAN, J.-S.; CHEN, S.-M.; NGUYEN, N. (Ed.). **Computational Collective Intelligence. Technologies and Applications**. Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6423). p. 63–70. ISBN 978-3-642-16695-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-16696-9_8>.

YOU, X.; TANG, Y.-Y. Wavelet-based approach to character skeleton. **Image Processing, IEEE Transactions on**, v. 16, n. 5, p. 1220–1231, May 2007. ISSN 1057-7149.

ZHANG, Y.; ZHANG, C. A new algorithm for character segmentation of license plate. In: **Intelligent Vehicles Symposium, 2003. Proceedings. IEEE**. [S.l.: s.n.], 2003. p. 106–109.