INSTITUTO DE COMPUTAÇÃO - UNICAMP

Tarefa 01 MO445 - Detecção de Placa de Carro

Marcos Felipe de Menezes Mota - 211893
24 de outubro de 2018

1 INTRODUÇÃO

Análise de imagens é o processo de extrair informação e métricas relevantes de imagens digitais. Para exercício do processo usado na área de análise de imagens, este documento relata a aplicação dessa área para o problema de detecção de placas de carro. Ou seja, a entrada é uma imagem, neste caso cinza, de um carro e a saída desejada uma imagem com uma retângulo delimitando onde está a placa do carro de acordo com o algoritmo de detecção.

A abordagem usada para obter o resultado desejado foi o uso de classificadores fracos. Dado um conjunto de características um classificador fraco, é um classificador binário, que define um limiar onde qualquer valor abaixo desse limiar é considerado uma classe e acima do limiar outra classe. No contexto do problema das placas a definição das características é dada por um conjunto de kernels de convolução e o classificador fraco determina se um pixel da imagem é um pixel de placa ou não [1]. Apesar de ser uma das técnica mais simples de análise de imagem, essa abordagem pode ser relacionada com rede neurais convolucionais (CNNs). CNN é a técnica mais usada atualmente para problemas de análise de imagem e seu poder de expressividade leva a crer que todas as outras técnicas de análise não serão mais necessárias. No entanto, CNNs podem ser vistas como uma sequência de aplicação de classificadores fracos, onde o conjunto de kernels de convolução são aprendidos usados técnicas de otimização como gradiente descendente. Tal relação foi guia para nossa implementação. [1]

Foi implementado um classificador fraco para um conjunto de kernels de Sobel e também para kernels aleatórios, gerados a partir de um script. Foi observado que a técnica aplicada retornou bons resultados tanto no conjunto de dados de treino como nos de teste, mas sua performance cai ao utilizar kernels aleatórios. A implementação de convolução utilizando

[1] O código e as imagens dos resultados podem ser acessadas em https://github.com/marcosfmmota/MO445-Code

multiplicação de matrizes, que permitiria otimizações e paralelização, acabou ficando incompleta. Nas seções a seguir será detalhado a implementação, os resultados obtidos e uma conclusão com as lições aprendidas com esse trabalho.

2 Técnicas e Dificuldades Práticas

Na Figura 2.1 temos o esquema da interpretação de um classificador fraco como uma rede convolucional com apenas uma camada. Inicialmente uma imagem cinza é dada como entrada para uma camada de rede convolucional. Uma rede convolucional em geral consiste dos seguintes passos [2]:

- Convolução: um pixel ij, e seus vizinhos definidos em uma relação de adjacência, são multiplicados por pesos (kernels) para definir o valor do seu brilho.
- ReLU: a função $max(0,x)$ é aplicada nos pixels da imagem. Essa função não linear ajuda a valorizar ativações dos kernels, além disso facilita o aprendizado em algoritmos de otimização.
- Pooling: novamente utilizando uma relação de adjacência, escolhemos o valor do pixel. A operação de pooling mais comum é a max-pooling onde o valor do pixel é substituído pelo pixel de maior brilho na adjacência. Além disso, é comum definir um parâmetro de subamostragem chamado stride que faz pooling ignorando os próximos pixels.

Esse passos estão na implementação com a única diferença que o valor de stride é 0 pois como a rede não é profunda não a necessidade de fazer subamostragem. Além disso, após um max-pooling é feito um min-pooling, nesse caso a escolha é do mínimo na adjacência, pois foi constatado empiricamente que tal operação realça melhor componentes de placa na imagem. O número de kernels e seus valores são dados na entrada do programa. A saída da camada de convolução é uma imagem multibanda com cada banda sendo a ativação de um kernel para a imagem.

No entanto, não sabemos quão bom é determinado kernel para detecção de placa. Para solucionar isso, podemos definir pesos (wj na figura) cada kernel e assim valorizar aqueles que melhor realçam placa. Com os pesos adequados, podemos combinar as bandas e definir um limiar (Circulo T na figura) que melhor captura os componentes desejados. Depois de encontrado o limiar podemos aplicar um pós-processamento para definir uma caixa delimitadora em volta da placa.

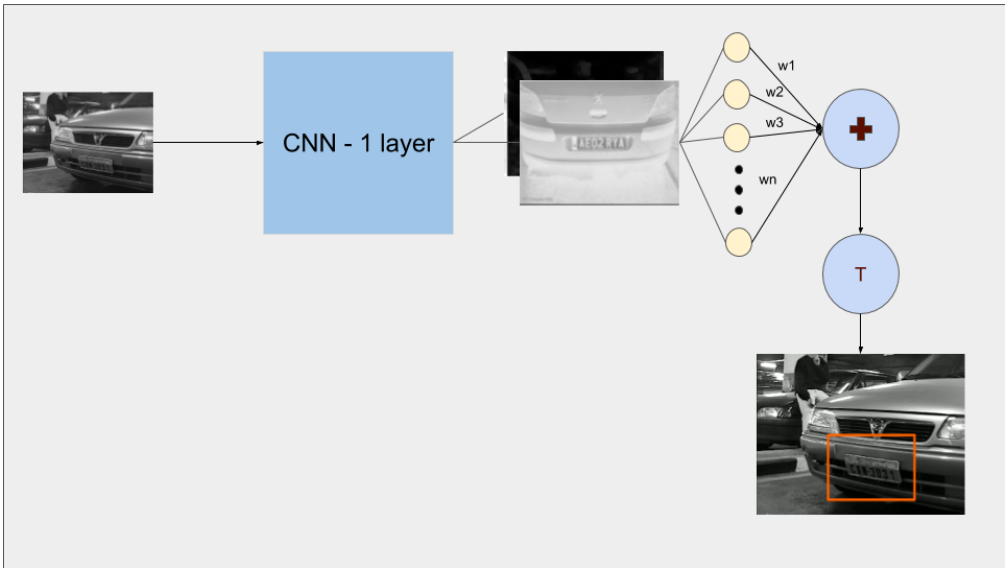


Figura 2.1: Arquitetura do algoritmo de classificação utilizado.

Na Figura 2.2 podemos ver um exemplo onde a saída do classificador define uma boa região de placa.

Figura 2.2: Resultado obtido com o algoritmo.

2.1 Aprendendo parâmetros

Para obter bons resultados como o mostrado na Figura 2.2, os pesos foram aprendidos utilizando aprendizado supervisionado, ou seja, além da imagem do carro foi utilizado um conjunto de dados de máscaras indicado a região da placas. Os pesos foram calculados da seguinte maneira:

- As imagens passam pela camada de CNN.
- Para cada banda de todas as imagens é calculado um limiar pela fórmula $e_{ij}(T_j) = \alpha N^0_{ij} + \beta N^1_{ij}$. e_{ij} é o erro para um determinado limiar para uma banda, onde N^0_{ij} é

o número de pixels de fundo classificados como placa e N^1_{ij} é o número de pixels de placa classificados como fundo. Escolhemos o menor e_{ij} fazendo busca exaustiva nos possíveis valores de limiar. Após encontrar o melhor limiar para cada banda os pesos são definidos pela fórmula $w_j = 1 - \frac{e_j}{\sum_{l=1}^{k} e_l}$.

- Encontrado os pesos, devemos definir o limiar real para o classificador fraco. Para isso os valores de cada banda são recalculados agora multiplicando o valor pelo peso do kernel. Depois disso as bandas são combinadas e o erro para um limiar é calculado de forma análoga a anterior. Percorrendo todo espaço de limiares é possível encontrar o melhor para os dados de treino.

2.2 Dificuldades

A principal dificuldade encontrada foi garantir a corretude da função de encontrar os pesos. A abordagem inicial para encontrar os pesos para cada banda foi imaginar cada banda como uma imagem de ativação e criar essa imagem utilizando `iftCreateImageFromBuffer` no entanto uma banda de uma `iftMImage` é uma array de floats e o buffer esperado a `iftCreateImageFromBuffer` é inteiro. Nessa ocasião C faz cast, no entanto tal abordagem acabou se mostrando um fruto de problemas de memória, que foi identificado posteriormente com a ferramenta *Valgrind*. Como `iftBand` é um wrapper para um vetor de floats foi utilizado o tipo `iftBand` mesmo.

Outra dificuldade encontrada na função de pesos foi delimitar um bom valor para α e β, pois no início, como não há pesos treinados, erros do tipo 1 (falsos positivos) são ordens de magnitude maior que tipo 2 (falso negativos). Por questão numérica α deve ser uma fração do valor de píxels de erro tipo 1, no início estava aumentado apenas as ordens de magnitude em β. Após encontrar esses valores, a implementação gerou bons resultados para conjuntos de dados menores que utilizei para debugging.

Um fator importante que foi negligenciado inicialmente foi alocação de memória. O resultados estavam promissores, mas ao aplicar aos folds (2xK-fold) de 100 imagens estabelecidos, havia um grande uso de memória. Isso fez com que o processo fosse terminado prematuramente pelo sistema operacional ou o computador parava de funcionar. Com isso foi visto a importância de usar `iftDestroyImage` para imagens auxiliares.

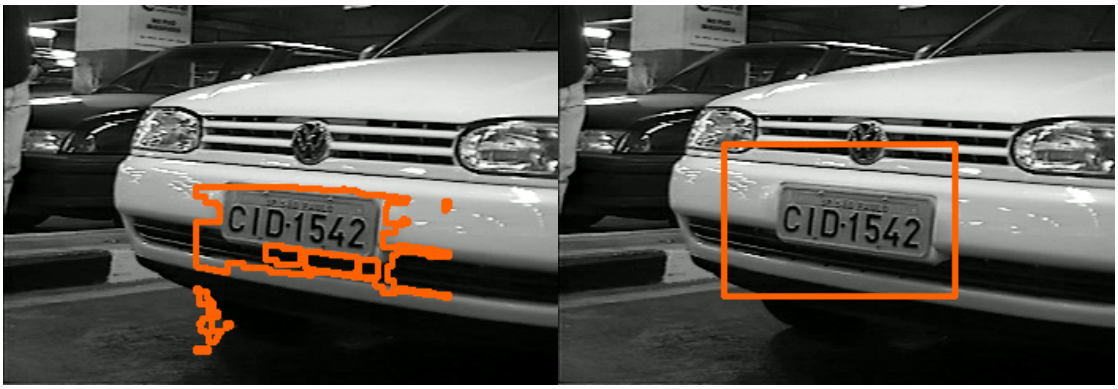
Além dessas dificuldades, houve apenas outras poucas relacionadas ao uso incorreto de certas funcionalidade da biblioteca de IFT.

3 Resultados e Experimentos

Para garantir bons resultados foram feitos vários testes nos parâmetros que influenciam a definição do limiar e na camada CNN.

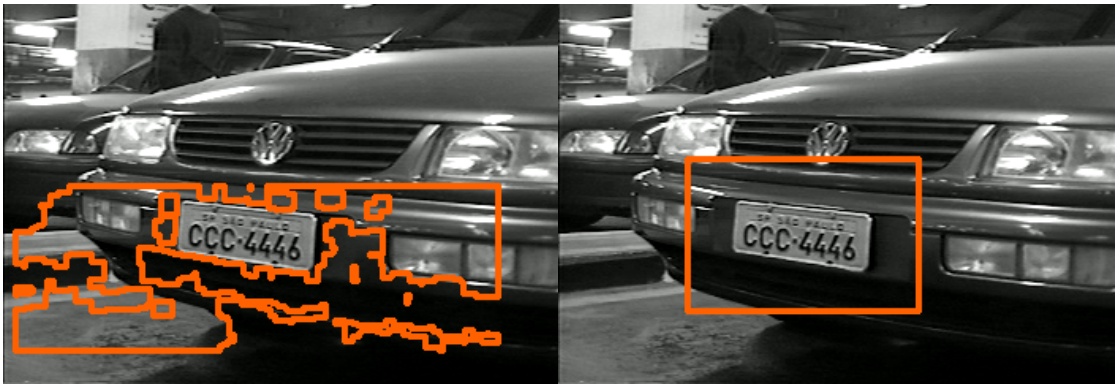
Um dos primeiros testes feitos foi a influência do pós-processamento na definição da caixa delimitadora. Na Figura 3.1 e 3.2 podemos ver quais componentes estão ativados para placa antes de ser aplicado pós-processamento. Como pode ser visto esses componentes podem estar bem próximos da região de placa, o que definiria quase um contorno do objeto, como

podem estar bem espalhadas. O pós-processamento faz uma série de erosões e dilatações para unir componentes próximos, definir qual componente é o mais próximo da média e definir a delimitação em volta dela. Na Figura 3.1 vemos o lado negativo dessa abordagem, o componente ativado está muito próximo da placa e com outro componente muito próximo também, isso faz com que a caixa delimitadora superestime a localização da placa e desloque um pouco para onde estava o outro componente. No entanto, como vemos em 3.2 o pós-processamento é adequado para quando se é detectado vários elementos além da placa, sendo assim uma alternativa segura de delimitar o objeto de interesse. Algo que influência a granularidade das componentes é o min-pooling e esse como foi discutido entre os alunos fornece uma melhor performance quando alterado para uma adjacência de tamanho 5x5.



(a) Imagem sem pós-processamento (b) Imagem com pós-processamento

Figura 3.1: Quando a componente encontrada é boa o pós-processamento pode ser influenciado por componentes próximos.



(a) Imagem sem pós-processamento (b) Imagem com pós-processamento

Figura 3.2: Quando a componente encontrada não contorna bem apenas a placa, o pós-processamento ainda é invariante.

Como foi falado na subseção de dificuldades, a escolha de α e β foi essencial para a qua-

lidade dos valores encontrados. Foram testados diferentes ordens de magnitude entre α e β. Como muito dos valores inadequados acabavam gerando pesos com valor NaN não há correspondente em imagem. A ordem de magnitude entre α e β foram 1000, 100, 10 e 1, com β sendo maior que α. Após esses testes, foi identificado que os valores de β devem ficar entre 50 e 10 de alpha, onde o melhor valor para identificar o limiar final é $\beta = 10\alpha$.

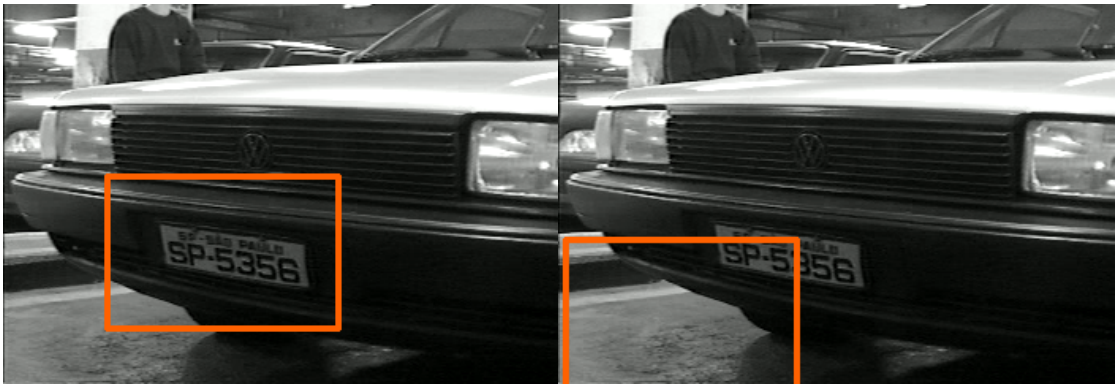
Após definido e testado os parâmetros do algoritmo, foi implementado um script que gera kernels. Os kernels iniciais usavam filtros Sobel ou seja detectores de borda em alguma direção. A hipótese seria que filtros aleatórios com norma 1 poderiam ressaltar características além de borda que facilitaria a detecção de placa. Abaixo está o trecho principal do script em Python que gera um banco de kernels e depois escreve em um arquivo.

```
from random import gauss

#A parte da implementação referente a escrita em arquivo foi omitida.
def generate_kernel_bank(filename="rand-kernel-bank.txt", nbands=1, kwidth=3,  \
    kheight=3, nkernels=8):

    kbank = []
    dim = kwidth*kheight
    for i in range(nkernels):
        kernel = [gauss(0, 1) for i in range(dim)]
        mag = sum(x**2 for x in kernel) ** .5
        kernel = [x/mag for x in kernel]
        bias = gauss(0,1)
        kernel.append(bias)
        kbank.append(kernel)
```

Ao ver o resultados dos kernels aleatórios, a hipótese não se confirmou. O resultado piorou e não houve um caso onde os kernels anteriores falharam e os aleatórios não. Na Figura 3.3 podemos ver um caso de falha da solução aleatória. Pela área delimitada na figura e em outros casos parece que o kernel acabou realçando o meio-fio confundindo o classificador mais vezes, logo piorando o resultado.



(a) Kernels de Sobel (b) Kernels aleatórios

Figura 3.3: Utilizar kernels aleatório comete erros em resultados antes corretos.

A Figura 3.4 mostra um exemplo onde o classificador erra. Essa é uma imagem difícil, pois o farol do carro tem muito brilho e um formato retangular parecido com a placa. Dessa forma qualquer kernel baseado em borda vai reforçar o farol e não a placa. Para contornar tal problema que foi sugerido utilizar kernels aleatórios, remover essa dependência de bordas, no entanto essa solução não funcinou como o esperado.

Figura 3.4: Caption

Para avaliar de maneira quantitativa a performance do classificador fraco foi computado algumas métricas para os melhores parâmetros nos conjuntos de teste. As métricas estão resumidas na Tabela 3.1.

Fold	1	2	3	4	5
Erro Sobel Kernels	9676.70	9369.71	9334.37	10354.46	9457.60
Erro Random Kernels	14197.31	13567.57	14083.39	15028.84	13489.34
Erros em Encontrar Placa Sobel	2	2	2	5	4
Média Sobel					9638.57
Desvio Padrão Sobel					377.23
Média Random					14073.29
Desvio Padrão Random					552.32

Tabela 3.1: Resultados no conjunto de teste para kernels de Sobel e aleatório e seus respectivas média e desvio padrão.

O erro médio nos folds de teste foram computados da mesma forma que o erro médio para erro dos limiares no conjunto de treino, so mudando o fato que o erro foi computado para apenas um limiar, o que foi encontrado no treino. A tabela mostra esse erro para cada fold tanto utilizando os kernels de Sobel e kernels aleatórios. Foi constatado que os kernels aleatórios não corresponderam as expectativas, pois seu erro médio é de 14073.29 e com desvio padrão de 552.32 em contraste com o bom resultado de 9638.57 de erro médio com desvio padrão de 377.23 para kernels de Sobel. Isso mostra que os kernels definidos erram menos e são mais consistentes no teste. Para finalizar foi computado para cada fold de teste quantas imagens não identificaram corretamente o local de placa. Somando a linha 3 da Tabela 3.1 e sabendo que cada fold de teste contém 100 imagens, temos o valor da precisão do classificador fraco.

$$Precisão = \frac{500 - 15}{500} = 0.97$$

Portanto a solução implementada acerta o local da placa de um carro 97% das vezes. Esse é um valor muito bom para a técnica utilizada e para o problema em questão, mesmo o conjunto de dados estar em condições ideais e com boa qualidade amostral.

4 Conclusão

O processo de padrão em análise de imagem; detecção, amostragem, caracterização e classificação, foram utilizadas nesse processo, apesar desses passos serem simplificados em um classificador fraco [1]. Com isso, foi obtido um entendimento real dos possíveis problemas encontrados durante esse processo, incluindo os de nível mais baixo como estruturas de dados e acesso de memória e disco. Esse são fatores importantes no mundo real. Do ponto de vista teórico, o paralelo entre classificadores fracos e uma rede convolucional foi muito importante para o entendimento de redes convolucionais, pois tira um pouco do mistério que se coloca sobre essa técnica, e também colocar as técnicas clássicas de análise de imagem no contexto atual. O problema, detecção de placas de carro, é interessante e se mostrou adequado para a técnica utilizada. A precisão obtida nos conjuntos de teste foram altas, provando que com um bom conjunto de extratores de características, nesse caso os kernels, técnicas simples de classificação podem ser suficientes. Isso também mostra o interesse e as vantagens em se utilizar técnicas que tentam descobrir esses extratores automaticamente.

REFERÊNCIAS

https://www.ic.unicamp.br/ afalcao/mo445/fundamentals-of-image-analysis.pdf
http://cs231n.github.io/convolutional-networks/