Relatório Análise de Imagens - Detecção de Placas de Carro

Felipe Pereira - 263808

17 de janeiro de 2020

Resumo

Este relatório refere-se aos trabalhos da disciplina Análise de Imagens, ministrada em 2020 pelo professor Alexandre Falcão, apresentando os resultados obtidos ao implementar um pipeline de análise de imagens de carros com o objetivo de identificar as placas destes.

Objetivo

Esta primeira tarefa consiste no início do pipeline. Nela iremos trabalhar o dataset de imagens de carros. Este dataset terá cada imagem dividida em patches, aumentando assim a quantidade de imagens disponíveis.

Cada patch deverá ter W × H pixel e uma movimentação de (D x , D y) pixels. Cada patch, que originalmente é uma imagem em grayscale, deverá ser transformado e pseudo-colorizado em uma imagem RGB (Red-Green-Blue) e depois em Y-Cb-Cr (ou LAB).

Após estas transformações, deverá ser aplicado um processo de Batch Normalization que transformará cada image, Î em uma imagem Ĵ i = (D I , J i):

$$J_{ij}(p) = \frac{I_{ij}(p) - \mu_j(p)}{\sigma_j(p)}, \quad (1)$$

$$\mu_j(p) = \frac{1}{N}\sum_{i=1}^{N} I_{ij}(p), \quad (2)$$

$$\sigma_j(p) = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N} (I_{ij}(p) - \mu_j(p))^2}, \quad (3)$$

Figura 01
Fórmulas de Batch Normalization

Depois deste processo, deverá ser feito o processo de criação dos Random Kernels. Deverão ser criados b random kernels com tamanho w x h, w << W e h << H. Também deverão ser analisadas as diferenças de forçar e não forçar estes kernels a terem média igual a zero.

Estes kernels serão utilizados para a realização de convoluções nos patches. Além dos kernels criados de maneira aleatória, será testado um kernel Sobel para comparação. Por fim, executaremos uma função de ativação ReLu para chegar ao resultado final.

Dataset Composition

A primeira etapa deste estudo consistiu em dividir as imagens em patches. Estes patches foram divididos de acordo com o stride definido. Foram testados strides de 1,5,10,15 e 20 pixels. O stride de 15 pixels foi escolhido por ser o que melhor consegue separar adequadamente em um único patch as placas dos carros:

Imagem orig_0200.png
Patch 152

Imagem orig_0200.png
Patch 153

Cada patch em GRAYSCALE, foi transformado em BGR para enfim ser convertida em Y-Cb-CR:

Imagem orig_0200.png
Patch 152 em Y-Cb-Cr

Imagem orig_0200.png
Patch 153 em Y-Cb-Cr

Batch Normalization

Tirando a média dos valores, temos uma imagem parcialmente modificada:

Imagem orig_0200.png
Patch 152
Com média em zero

Cada imagem acima representa um canal da imagem. Apenas o primeiro canal possui algo que pode ser detectado através do olho humano e, mesmo assim, com auxílio computacional de Gamma adicional. Os outros dois canais normalizados não apresentam elementos visualmente perceptíveis a olho nu.

Convoluções

Foram criados 3 kernels aleatórios e 1 kernel sobel. Primeiro serão analisados os resultados dos kernels sem forçar com que a média dos mesmos fosse zero.

Resultado da convolução com os kernels sem a média igual a zero:

Imagem orig_0200.png
Patch 153
Convolução com filtro aleatório

Este kernel não ressaltou nenhum aspecto importante da imagem, causando, inclusive, perda de informação importante (o número da placa).

Resultado da convolução com os kernels com a média igual a zero:

Imagem orig_0200.png
Patch 152
Convolução com filtro aleatório e média zero

Este kernel ressaltou os relevos presentes na imagem.

Também é interessante comparar os resultados acima com os resultados da utilização de um filtro Sobel:

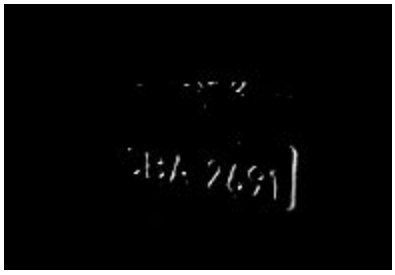

Imagem orig_0200.png
Patch 153
Convolução com filtro Sobel

Já o filtro sobel conseguiu extrair os relevos verticais, mostrando o relevo da placa e dos números e letras da placa.

ReLU

Nesta fase do pipeline aplicamos uma função de ReLu no resultado da convolução:

$$A_{ij}(p) = \max\{0, C_{ij}(p)\}.$$

Figura 02
Fórmula de Max Pooling

Vamos aplicar o ReLu ao resultado da convolução do filtro aleatório com média zero:

Imagem orig_0200.png
Patch 152
Relu com filtro filtro aleatório e média em zero

Vamos aplicar o ReLu ao resultado da convolução do filtro Sobel:

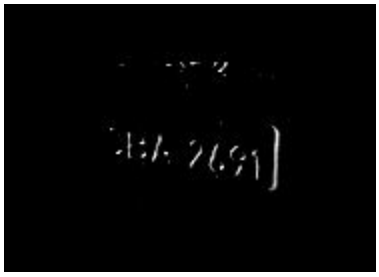

Imagem orig_0200.png
Patch 153
Convolução com filtro Sobel

Pooling

As operações de Pooling são responsáveis pelo processamento de imagem que, antes do advento das redes neurais, eram normalmente conhecidas pelos nomes de dilatação seguida de sub-amostragem (no caso do Max Pooling).

A operação de Max Pooling é responsável por verificar o valor de pixel mais alto em uma dada relação de adjacência, normalmente processada através de um kernel deslizante sobre a imagem. A definição teórica de Max Pooling é dada por:

$$P_i(p) = \max_{q \in \mathcal{A}(p)} \{R_i(q)\},$$

Fórmula de Max Pooling

Ao aplicar a operação de Max Pooling após a operação de ReLu, temos o seguinte resultado:

Imagem orig_0200.png
Patch 152
Max Pooling depois de ReLu

Outra possibilidade, é executar uma função de Average Pooling, definida por:

$$P_i(p) = \frac{1}{|\mathcal{A}(p)|} \sum_{q \in \mathcal{A}(p)} \{R_i(q)\},$$

Figura 03
Fórmula de Average Pooling

A operação de Average Pooling é relativamente parecida com a de Max Pooling em sua forma de atuação, tendo como diferença principal, a execução da média dos pixels da relação de adjacência. Temos como resultado desta operação:

Imagem orig_0200.png
Patch 152
Average Pooling depois de ReLu

Super Pixels

Os chamados “Super Pixels” são imagens definidas em áreas e, dentro destas áreas, temos que apenas um pixel se sobressai aos demais, daí o nome de “Super Pixel”. Esta técnica simplifica a imagem e tenta extrair semânticas mais simples (com uma diversidade menor de valores de pixels) em comparação com a imagem original.

A dificuldade no processamento dos Super Pixels está, exatamente, em fazer a simplificação dos pixels, sem perder informações relevantes, tais como: objetos, contexto e semântica.

Seguem abaixo, exemplos da mesma imagem dividida em uma quantidade diferente de Super Pixels. Isto demonstra que, escolher a quantidade correta de super pixels, é uma tarefa dependente da imagem e altamente correlacionada com o objetivo que se quer atingir (em nosso caso, identificar a placa). Isto torna esta técnica um pouco mais difícil de ser escalada de maneira automatizada para centenas ou milhares de imagens diferentes:

Figura 02
Imagem com 10 super pixels

Figura 03
Imagem com 30 super pixels

Figura 04
Imagem com 50 super pixels

haar-like

Existem diversas formas de extrair características das imagens e a haar-like é mais uma delas. Ela é muito utilizada para a extração de características importantes em imagens de rostos, dado que sua natureza permite a extração de características de identificação importantes como: sobrancelhas, nariz e boca.

Para cada patch da figura, foram extraídas as haar features de apenas 1 canal que serão, posteriormente, passadas para um classificador (no caso, Support Vector Classifier), para que tenhamos um pipeline completo de detecção de placas.

Dado que as haar features são calculadas através da integral do filtro, em cada conjunto de pixels analisados, seu processamento é custoso. Por isso, foram processadas apenas algumas imagens.

A expectativa é de que as placas possam ser facilmente detectadas, pois possuem um tamanho padrão e letras em seu interior. Com isso, a expectativa na modelagem é de que o modelo treinado com as haar features apresente um bom resultado. O lado negativo é que muitas características são extraídas, para cada canal, de cada path, são mais de 5 milhões de pixels, somados após o flatten da operação.

Apesar de serem computacionalmente custosas, as características extraídas conseguiram treinar um modelo muito bom, obtendo ótimas métricas. Ainda que o modelo possa ter entrado

em “overfit”, acreditamos que o desempenho do mesmo continuaria bom com uma quantidade maior de registros (abaixo de 100%, claro):

	precision	recall	f1-score	support
not_plate	1.00	1.00	1.00	1
plate	1.00	1.00	1.00	2
accuracy			1.00	3
macro avg	1.00	1.00	1.00	3
weighted avg	1.00	1.00	1.00	3

Figura 05
Resultados do modelo treinado com haar features

FLIM

Geralmente utilizamos as imagens para treinar os modelos. Porém, foi proposta, pelo Profº Alexandre Falcão, uma metodologia para extrair características atráves da marcação de imagens, originando-se assim a metodologia FLIM (Feature Learning from Image Markers).

Esta metodologia foi codificada em python através da bilbioteca de mesmo nome (FLIM), criada pelo orientando Italos Estilon.

Esta biblioteca permite que a metodologia FLIM seja aplicada, utilizando-se de marcações manuais em imagens para que as características extraídas sejam melhoradas, de acordo com o objetivo, para que isso resulte em características melhores que, por sua vez, deverão proporcionar modelos de classificação mais acurados.
Através do uso da ferramenta de “annotation”, foram criados markers em 2 patches. Abaixo segue o exemplo de como criar as marcações:

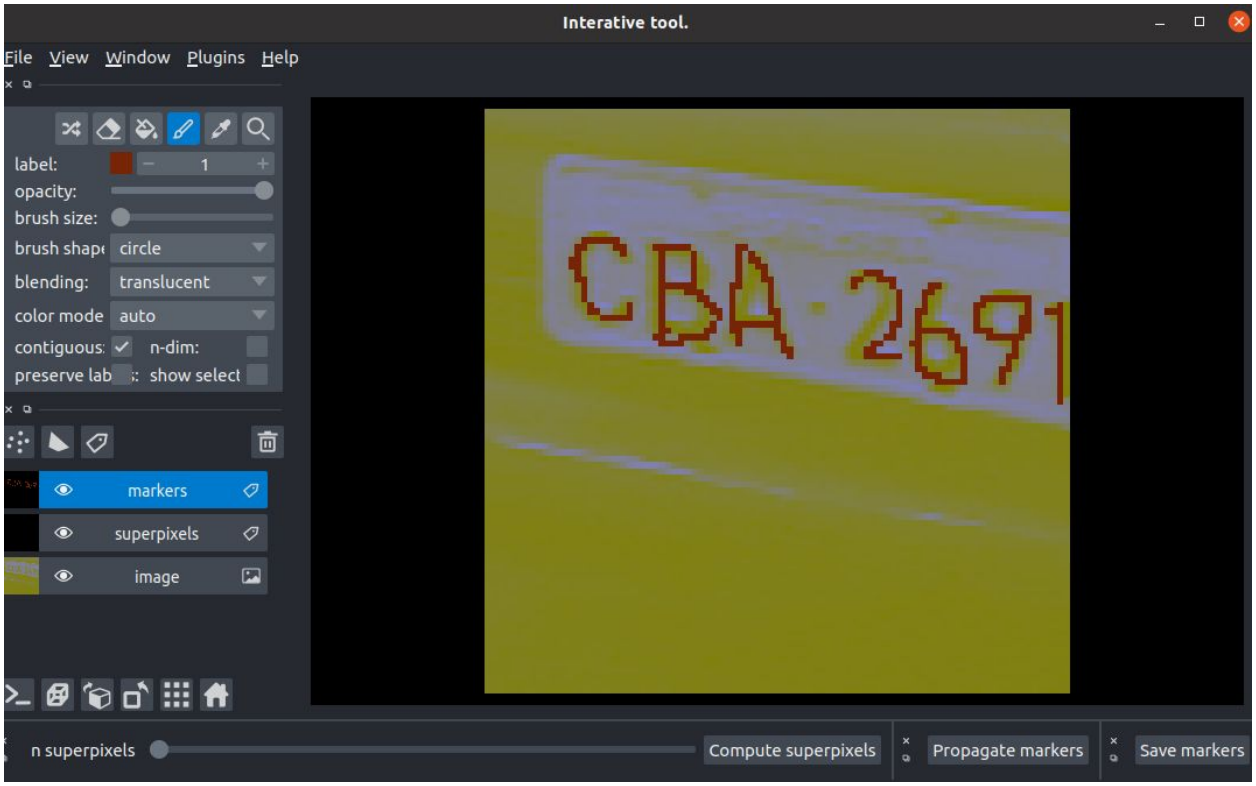

Figura 06
Marcações sendo feitas no patch

A ferramenta gera um arquivo com o padrão esperado pela função “load_images_and_markers”:

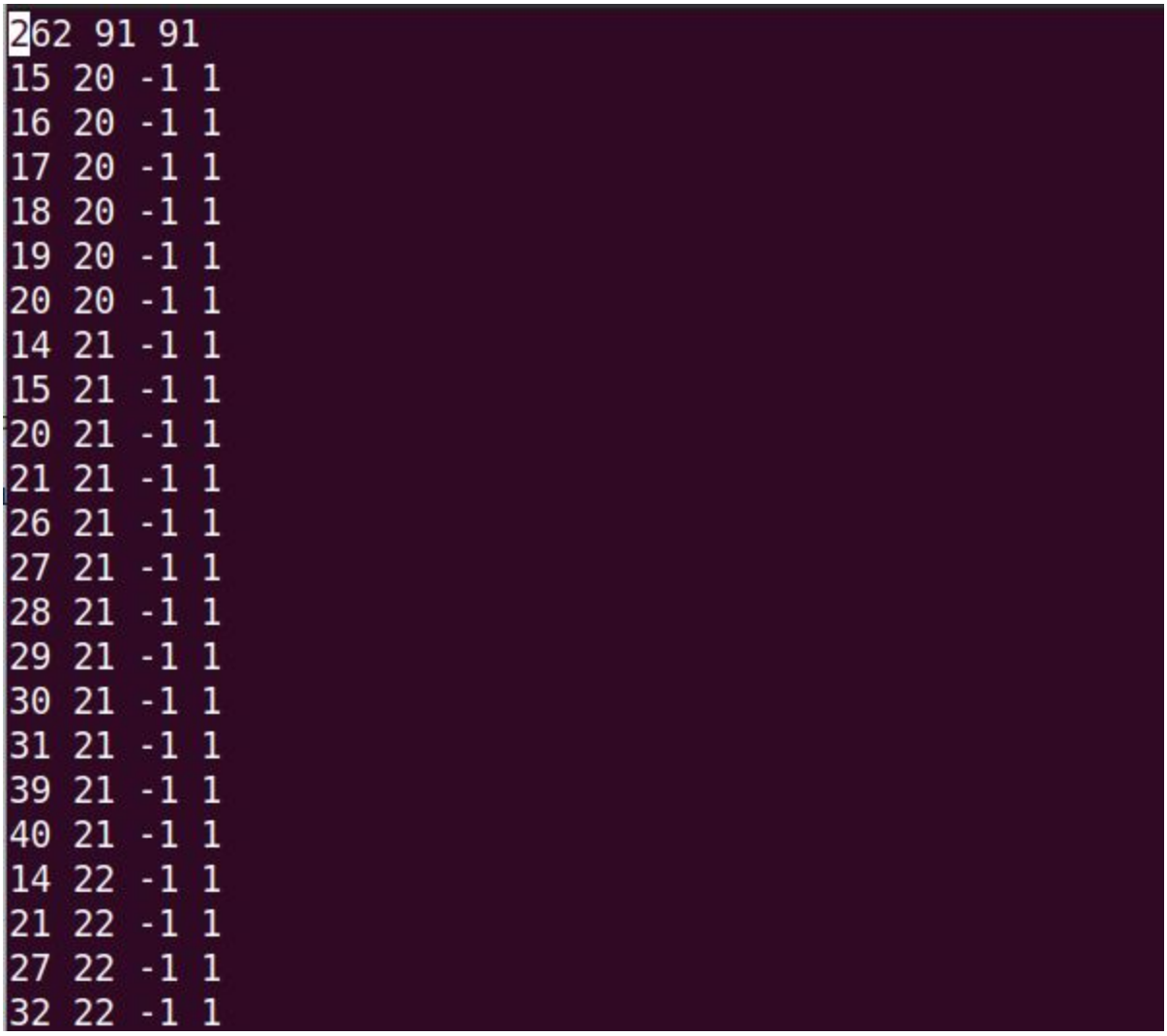

Figura 07
Porção de um arquivo de marcações

Dado que esta técnica combina a marcação de imagens com a detecção de características, esperamos que ela tenha um alto desempenho na fase de modelagem e classificação.

Modelagem

Conforme proposto no exercício, foi usada a biblioteca de Support Vector Machine presente no pacote scikit-learn do Python. Utilizei a LinearSVC com $C=10^2$, conforme proposto.

Nosso propósito nesta parte do processo é, menos o de otimizar algum modelo específico, mas sim o de verificar a influência das características (*features*) extraídas das imagens, comparadas através do desempenho de um mesmo modelo, treinado com os mesmos parâmetros.

Deste modo, excluímos a influência do modelo e podemos analisar (se não de maneira direta, ao menos de maneira indireta), como cada conjunto de extração de características afeta a tarefa de classificação de imagens.

Foram classificadas como classes positivas as imagens com mais de 50% (maior igual à) da placa. As outras (menor que 50%), foram consideradas classes negativas.

Foram testadas as seguintes técnicas de extração:

- **Método 01**: Patching da imagem original. Nenhuma transformação foi realizada, além de se realizar o patching da imagem original**.**
- **Método 02**: Convolução, compreendendo um *pipeline* que inclui *Patching*, *Batch Normalization*, ReLu e *Max Pooling. Foi utilizado um kernel gerado de maneira aleatória com média em zero.*
- **Método 03**: Convolução, compreendendo um *pipeline* que inclui *Patching*, *Batch Normalization*, ReLu e *Max Pooling.* Foi utilizado um kernel Sobel.
- **Método 04**: haar-like. Foram extraídas características de haar. Foram usados apenas 10 registros devido ao custo computacional.
- **Método 05**: FLIM. Técnica proposta pelo professor para este exercício. Foram utilizados quatro registros, dois da classe positiva e dois da classe negativa.

Os modelos foram treinados com os dados enviados através de um “pandas DataFrame”. O DataFrame consiste em um objeto de manipulação de dados com 2 dimensões (linhas e colunas).

Cada linha representa uma imagem (patch) e as colunas criadas foram as seguintes:

-Patch_Number: O número do patch dentro da imagem. Esta informação não é passada para o treinamento do modelo, está lá apenas para facilitar a verificação dos dados, quando é necessário validar manualmente alguma informação.

-Class: Existem duas classes. Zero (0) significa que não há placa (< 50%). Um (1) significa que há placa no patch (>= 50%).

-Plate_Proportion: A proporção da placa presente no patch. Varia de 0 a 1. Esta informação não é passada para o modelo, está presente apenas para validação de dados manual.

-Colunas numéricas (de 0 até a quantidade de pixels): São o resultado do flatten das imagens (patches). Mais especificamente, é o flatten das features extraídas, de acordo com cada método de extração. Cada método de extração gera um DataFrame diferente que serve como insumo para o treinamento dos diferentes modelos.

Exemplo de DataFrame:

	patch_number	class	plate_proportion	0	1	2	3	4	5	6	...	8271	8272	8273	8274	8275	8276	8277	8278	8279	8280
0	1	0	0.0	87	91	95	93	96	100	103	...	98.0	99.0	99.0	101.0	92.0	77.0	72.0	78.0	72.0	53.0
1	2	0	0.0	155	167	171	166	171	168	162	...	41.0	40.0	38.0	38.0	37.0	38.0	37.0	38.0	38.0	35.0
2	3	0	0.0	162	91	54	67	77	80	82	...	36.0	39.0	39.0	39.0	37.0	38.0	43.0	66.0	87.0	69.0
3	4	0	0.0	134	98	105	110	101	94	90	...	158.0	156.0	153.0	156.0	160.0	158.0	155.0	158.0	157.0	153.0
4	5	0	0.0	221	217	212	194	197	195	189	...	218.0	218.0	218.0	219.0	219.0	218.0	218.0	219.0	218.0	218.0

5 rows × 8284 columns

Figura 08
Exemplo de DataFrame com as características extraídas

Os resultados alcançados foram:

Método	**Qtde registros (Treino + Teste)**	**Classe**	**Precisão**	**Recall**	**F1**	**Acurácia**
Patching	228	não_placa	88%	100%	94%	89%
		placa	100%	44%	62%	
Convolução (Random Kernel)	228	não_placa	89%	95%	92%	85%
		placa	33%	17%	22%	
Convolução (Sobel Kernel)	228	não_placa	85%	67%	75%	60%
		placa	7%	17%	10%	
haar	10	não_placa	100%	100%	100%	100%
		placa	100%	100%	100%	
FLIM	4	não_placa	100%	100%	100%	100%
		placa	100%	100%	100%	

Tabela 01
Métricas de classificação dos métodos utilizados

Conclusão

Existem diversos tipos de parâmetros que podem ser ajustados no pipeline. Indo desde a escolha do tipo de imagem a usar, tamanho dos patches, stride, formas de realizar a normalização, os valores a serem usados nos filtros, dentre outros.

Foi constatado que filtros com a média centrada em zero possuem um comportamento melhor do que filtros com valores completamente aleatórios. Além disto, foi constatado também que

filtros já conhecidos e testados, como o Sobel, possuem efeitos conhecidos e com ótimos resultados para os resultados.

Nosso pipeline foi incrementado com diversos tipos de técnicas para extração de características. Estas técnicas serviram para testarmos um modelo de classificação (SVM), que classifica cada imagem em duas classes:

- Possui placa
- Não possui placa

Através da análise do modelo, identificamos que as diferentes técnicas apresentam diferentes resultados e que, por se tratarem de imagens, o processamento adequado deve ser feito em máquinas com capacidade para isso. Não obstante, possuir o conhecimento de como estas técnicas funcionam, nos possibilita testá-las em diversos tipos de cenários, escolhendo aquelas que apresentarem o melhor desempenho em cada situação.