INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Sistema de Classificação de Imagens

Relatório

Luís Filipe Daio Fernandes

N.º 17186

2020

INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Sistema de Classificação de Imagens

Relatório

Elaborado por:

Luís Filipe Daio Fernandes

N.º 17186

2020

Índice

Índice de Figuras

Capítulo 1

Introdução

Este trabalho está inserido na unidade curricular de Linguagens de Programação e tem como objetivo o desenvolvimento de um sistema de reconhecimento de imagens, no conjunto de classes de objetos seguinte: $C = \{$*edificados, floresta, glaciar, montanha, mar, rua*$\}$.

O sistema foi desenvolvido usando a linguagem de Python, fazendo uso das bibliotecas OpenCV, Numpy, Os, Pickle, Mahotas e Scikit-learn. As imagens utilizadas no treino do modelo foram obtidas através do dataset[1] disponibilizado pela Intel para um concurso de classificação de imagens. A aplicação web aonde funciona o sistema foi desenvolvida com o pacote Flask e permite fazer upload de imagens com o formato PNG ou JPEG para posterior classificação.

O presente relatório encontra-se divido em três partes. Na primeira parte explica-se de uma forma teórica todo o processo que envolve a classificação das imagens. Na segunda parte mostra-se a utilização do sistema desenvolvido, assim como os resultados obtidos com algumas imagens de teste. Por fim na terceira parte é feita uma conclusão sobre o trabalho realizado e resultados obtidos, assim como uma proposta para melhoramento futuro.

Capítulo 2

Teoria

Neste capítulo é feita uma descrição do processo de classificação de imagens utilizado neste trabalho. Este processo divide-se nas seguintes etapas: aquisição das imagens(dataset); pré processamento das imagens, extracção das características, treino do modelo, classificação.

2.1 Aquisição das Imagens

O dataset disponibilizado para o desenvolvimento deste trabalho está organizado como mostra a Figura 2.1. Dentro da directoria "`seg_train`"estão cerca de 14 mil imagens para treinar o modelo, que por sua vez estão divididas em sub-pastas de acordo com a classe a que pertencem. Dentro da directoria "`seg_test`"estão cerca de 3 mil imagens para fazer a validação cruzada do modelo, e as imagens também estão divididas consoante as classes a que pertencem. Na directoria "`seg_pred`"estão cerca de sete mil imagens para efectuar a previsão, depois de o modelo estar treinado. Todas as imagens do dataset têm o tamanho de 150x150 pixeis.

2.2 Pré-Processamento

Nesta secção são descritos os procedimentos utilizados para pré processar as imagens de modo a melhorar e uniformizar a extracção das características das imagens, que é feita no passo seguinte. Este pré processamento consiste em redimensionar as imagens, seguido da aplicação do filtro Gaussiano [2] e por fim a obtenção dos contornos através do algoritmo Canny[3]. Como referido anteriormente todas as imagens do dataset têm o mesmo tamanho de 150x150 pixeis e assim sendo não há necessidade redimensionar. Caso se opta-se por aumentar o tamanho das imagens,

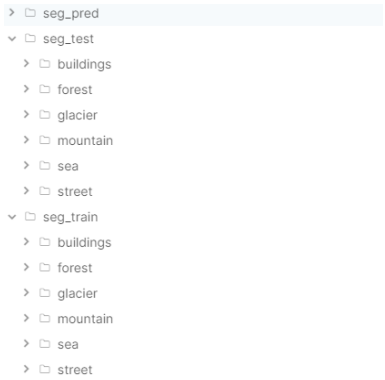

Figura 2.1: Estrutura do Dataset

estaria-se a perder a qualidade das mesmas, podendo dar resultado a resultados menos satisfatórios.

2.2.1 Filtro Gaussiano

Este filtro é utilizado para remover o ruído da imagem fazendo uso da função Gaussiana (também conhecida por distribuição normal na estatística). Para atingir este efeito o valor de um pixel da imagem original é multiplicado por todos os valores de um kernel Gaussiano(Figura 2.2) , que pode ser 3x3, 5x5, etc. Posteriormente todas essas multiplicações são somadas e o valor resultante dessa soma é então dividido pela soma de todos os valores do kernel. O resultado final corresponde ao valor do pixel na imagem resultante. Aplicando este processo a todos os pixeis da imagem obtém-se uma imagem suavizada. Quanto maior for o kernel, mais suave é a imagem resultante, assim como mais tempo demora, dado que são feitas mais operações.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figura 2.2: Kernel Gaussiano(5x5)

Com a aplicação da função GaussianBlur()[1] do pacote OpenCV é possível visu-

[1]Filtro Gaussiano. `https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html`

alizar os efeitos do filtro(Figura 2.3).

Figura 2.3: Exemplo da aplicação do filtro Gaussiano

2.2.2 Filtro *Canny*

O algoritmo Canny foi desenvolvido por John F. Canny em 1986 [3]. É composto pelas seguintes etapas [4]:

1. Aplicação do filtro Gaussiano;
2. Cálculo do gradiente(taxa de variação) de intensidade da imagem;
3. Supressão dos não-máximos;
4. Aplicação do duplo limite;
5. Identificação dos contornos por histerese.

Aplicação do filtro Gaussiano

Os resultados da aplicação do algoritmo podem ser altamente influenciados pela quantidade de ruído na imagem e assim sendo a aplicação do filtro Gaussiano faz parte do algoritmo.

Cálculo dos Gradientes

O cálculo da taxa de variação da intensidade detecta a intensidade e direção dos contornos da imagem usando operadores de deteção de contornos. Este efeito é atingido utilizando kernels Sobel [5].

Supressão dos Não-Máximos

Após a etapa anterior os contornos da imagem não estão uniformizados (uns são mais grossos do que outros). Ao aplicar supressão dos não-máximos é possível tornar os contornos mais finos.

Aplicação do Duplo Limite

Após a etapa anterior os contornos ainda não estão uniformizados, mas desta vez deve-se à intensidade dos pixeis: alguns estão mais claros do que outros. É aqui que entra a aplicação do duplo limite. São definidos dois limites de intensidade. Os pixeis com intensidade superior ao limite superior são considerados fortes; os pixeis com intensidade inferior ao limite inferior, são considerados irrelevantes; os pixeis com intensidade entre os dentro dos limites, são considerados pixeis fracos. Após esta etapa a imagem resultante é composta por pixeis pretos, cinzentos e brancos que correspondem aos pixeis irrelevantes, fracos e fortes, respetivamente.

Identificação dos Contornos por Histerese

Este último passo consiste em transformar os pixeis fracos em fortes, se e apenas se pelo menos um dos pixeis à volta daquele que se encontra a ser processado é forte.

A função Canny()[2] do pacote OpenCV permite aplicar a este algoritmo a imagens e obter uma imagem dos contornos, como mostra a Figura 2.4.

2.3 Extração das Características

Para a extração das características aplica-se o processo anterior a todas as imagens presentes na directoria "`seg_train`"guardando as mesmas e as classes a que pertencem (edificados, mar, floresta, etc.) nas respectivas listas, à medida que se vão sendo extraídas. No desenvolvimento deste trabalho optou-se por extrair apenas características relativas à textura da imagem, *haralick* [6], dado que após a aplicação do detetor de contornos Canny, a imagem passa a ter apenas um canal de cor, e para a extração de características relativas à cor, seriam necessários três (RGB). A extração das características é feita recorrendo à função features.haralick() da biblioteca Mahotas[7], que à semelhança do OpenCV é uma biblioteca de visão computacional e processamento de imagem. Após a extração de todas as características, as listas

[2]Detetor de contornos. `https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html`

Figura 2.4: Exemplo da aplicação do detector de contornos Canny.

contendo as mesmas são guardadas num ficheiro Pickle[3], para prevenir que se tenha de repetir este processo cada vez que se queira treinar o modelo.

2.4 Treino do Modelo

O Scikit-learn [8] é uma biblioteca de *machine learning* [4] de código aberto para Python. Após o teste dos vários modelos de oferecidos pelo Scikit-learn, o modelo escolhido para o desenvolvimento deste trabalho foi o MLP (Multi-layer Perceptron[5] pois apresenta uma boa relação taxa de acerto/tempo de execução.

Nesta etapa do processo carregam-se do disco as características das imagens extraídas na etapa anterior. Seguidamente, procede-se à codificação das classes em estudo recorrendo à função fit_transform() da classe Label_Encoder do Scikit-learn. Posteriormente, procede-se ao treino do modelo usando a função fit() do mesmo, tendo como parâmetros a lista de características e as codificações das classes. Por fim o modelo, já treinado, é serializado para o disco, recorrendo novamente a um ficheiro Pickle.

[3]Pacote de Python que permite a serialização e desserialização de objectos. `https://docs.python.org/3/library/pickle.html`

[4]Aprendizagem automática

[5]Algoritmo de *machine learning* supervisionado de retro-propagação.

2.5 Classificação

Para a classificação, repetem-se as etapas de pré-processamento e extracção das características que se quer testar. Seguidamente carrega-se o modelo treinado recorrendo à função loads() do pacote Pickle. Por fim obtém-se as previsões do modelo através da função predict_proba() do mesmo.

Capítulo 3

Realização Experimental

O sistema foi desenvolvido na linguagem Python com uso das bibliotecas mencionadas na introdução. O sistema utilizado para o efeito foi uma máquina virtual Ubuntu 18.04 com 8 Gb de memória RAM e com 3 dos 6 cores da máquina host que por sua vez é um ASUS Vvivobook com as seguintes características:

1. Processador: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
2. Memmória RAM: 16GB
3. Placa gráfica: NVIDIA GEFORCE GTX 1050 4GB
4. SO: Windows 10 Home 64bits

O sistema foi testado com algumas imagens presentes na directoria "`seg_test`". Os resultados foram os apresentados nas Figuras 3.1, 3.2, 3.3

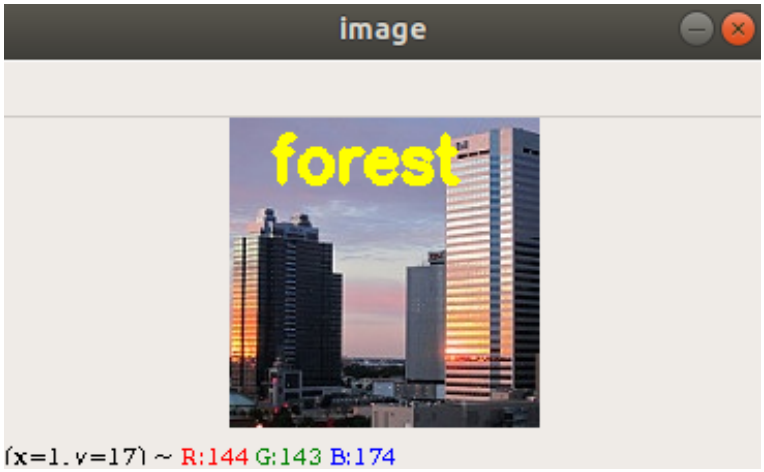

Figura 3.1: Resultados(1)

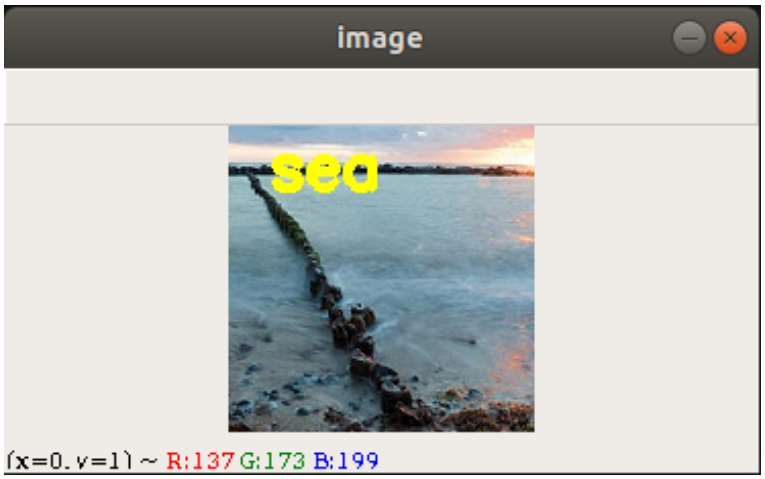

Figura 3.2: Resultados(2)

Figura 3.3: Resultados(3)

Capítulo 4

Conclusões

Bibliografia

[1] Intel image classification. Consultado em 2019/11/10. [*Online*]. Disponível: https://www.kaggle.com/puneet6060/intel-image-classification (citado na pág. 1)

[2] Gaussian blur. Consultado em 2020/01/9. [*Online*]. Disponível: https://en.wikipedia.org/wiki/Gaussian_blur (citado na pág. 3)

[3] Canny edge detector. Consultado em 2020/01/9. [*Online*]. Disponível: https://en.wikipedia.org/wiki/Canny_edge_detector#Gaussian_filter (citado nas págs. 3 e 5)

[4] (2019) Canny edge detection step by step in python. Consultado em 2020/01/9. [*Online*]. Disponível: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123 (citado na pág. 5)

[5] Sobel operator. Consultado em 2020/01/10. [*Online*]. Disponível: https://en.wikipedia.org/wiki/Sobel_operator (citado na pág. 5)

[6] Texture recognition using haralick texture and python. Consultado em 2019/12/5. [*Online*]. Disponível: https://gogul.dev/software/texture-recognition#what-is-a-texture (citado na pág. 6)

[7] Mahotas: Computer vision in python. Consultado em 2019/12/5. [*Online*]. Disponível: https://mahotas.readthedocs.io/en/latest/ (citado na pág. 6)

[8] Scikit.learn. Consultado em 2019/12/5. [*Online*]. Disponível: https://pt.wikipedia.org/wiki/Scikit-learn (citado na pág. 7)