Classificação de imagens de castas de videira com utilização de Deep Learning

Beatriz Lucas Neto e Moura Cardoso, nº 23195
Luís Maria Gama e Castro Moura Soares, nº 23224

2023

Índice

1. Introdução

No presente projecto, pretende-se identificar espécies de videira através da classificação de imagens das respectivas folhas.

A vinha é uma cultura com uma importância económica e social relevante em vários países do mundo. Em Portugal, a vinha ocupa 2.1% da área territorial e pela influência do sector vitivinícola na economia e sociedade portuguesa. O número de castas cultivadas mundialmente estima-se que se encontre entre as 5000 a 8000 variedades, sendo que muitas destas possuem muitas vezes mais do que uma designação comum.

Desta forma, a melhor identificação de castas através do uso de aprendizagem automatizada ajudará a:

- Melhor garantir aos viticultores e viveiristas que a casta adquirida é a pretendida, contribuindo para o controlo de fraudes e falsificações;
- Melhorar o processo de catalogação de novas variedades;
- Evitar e identificar casos de sinonímia e homonímia.
- Preservar os produtos e as variedades típicas dos países e regiões.

O desafio deste projecto é conseguir a identificação de espécies de videira através de classificação de imagens de apenas de um órgão das plantas: as folhas. A fim de obter uma precisão relativamente elevada na classificação destas imagens, será importante a utilização de um número elevado de imagens de folhas maturas das espécies consideradas sem manifestação de sintomas de ataques de pragas ou infecção de doenças, para fazer face à grande variabilidade de características foliares das videiras.

No presente projecto será utilizada uma base de dados disponibilizada no Kaggle pelo utilizador Murat Koklu, de nome *Grapevine Leaves Image Dataset* (https://www.kaggle.com/datasets/muratkokludataset/grapevine-leaves-image-dataset). A base de dados tem uma pontuação de 8.75, sendo que o único parâmetro onde tem nota negativa é na frequência de actualização do conjunto. A licença da *Creative Commons* associada é CC0 1.0 Universal, sendo o conjunto de dados de utilização pública sem quaisquer direitos de autor.

A base de dados está disponível para download, em formato zip, sendo constituída por 500 imagens de folhas de espécies de videira típicas da Turquia, nomeadamente, por 100 imagens de folhas das espécies Ak, Ala Idris, Büzgülü, Dimnit e Nazli.

O método e os passos propostos a realizar no presente projecto são: 1) *building a data pipeline,* 2) *pre-processing the images for deep learning*, 3) *creating a neural network classifier*, 4) *evaluating model performance* e 5) *saving the model for deployment.*

A avaliação da performance do modelo vai ser realizada com base nas seguintes métricas: precisão, exactidão e *recall*. Assim, será necessário estabelecer instâncias destas métricas e actualizá-las em função das previsões feitas.

2. Dados

Foram utilizadas 500 imagens com uma boa resolução e de elevada nitidez, uma vez que as amostras foram tratadas em ambiente controlado e fotografadas contra um fundo de alto contraste, tornado todas as características distintivas da folha mais evidentes. É de notar que o modelo, ao ser treinado exclusivamente com imagens altamente tratadas, serão necessárias imagens de igual qualidade para que este consiga classificar correctamente as imagens das folhas das castas.

Não foi necessária uma limpeza das imagens, visto que o conjunto de imagens utilizado já tinha sido previamente organizado em cinco pastas distintas, respectivas a cada casta estudada. O formato de ficheiro utilizado para o trabalho das imagens foi o .png por ser um dos dois formatos suportados pelo TensorFlow, sendo o outro .jpeg. Foi escolhido o formato .png em detrimento do formato .jpeg por ser este compatível com fundos transparentes, o que diminui a probabilidade do fundo interferir com a interpretação computacional dos atributos das folhas pelo rede neural. Para além desta característica, o formato .png permite uma menor compressão das imagens, o que resulta numa maior qualidade e quantidade de pixéis utilizáveis.

3. Métodos

Para a criação deste modelo foi utilizada a estrutura de aprendizagem TensorFlow por ser estruturalmente bastante completa nas suas ferramentas e compatibilidade com bibliotecas externas do ecossistema da linguagem de programação Python, como o Numpy e o *API* Keras (Abadi et al., 2016; Gulli & Pal, 2017; Harris et al., 2020). Foi também escolhido devido à elevada dimensão da comunidade de utilizadores activos, que promovem um ambiente de discussão e exemplos de utilização muito abrangentes, permitindo maior facilidade na sua utilização.

Das 500 imagens, retirou-se, de cada conjunto, uma imagem de cada casta para posterior demonstração. Assim, as pastas de cada casta possuíam um total de 495 imagens, 95 por casta, utilizadas para treinar, validar e testar o modelo.

Criaram-se 15 *arrays* de 32 imagens e um *array* de 15 imagens aleatórias, interpretados com valores do tipo *int32*. Como forma de agilizar a leitura dos dados guardados em memória, reduziu-se a dimensão do eixo dos x das imagens para ser compreendido entre 0 e 1 através do seguinte comando, “data = data.map(lambda x,y: (x/255, y))”.

Posteriormente, dividiram-se os 16 *arrays* em três conjuntos, “train”, “val” e “test”, para treino, validação e teste, respectivamente. Para treino são utilizados 70% dos *arrays*, 20% para validação e 10% mais 1% *array* para teste.

Antes de iniciar o treino do modelo é necessário preparar as transformações a aplicar nos dados. A ordem de aplicação dessas alterações implicou a utilização do método *Sequential* do *API* Keras que, como o nome indica, aplica de forma ordenada e sequencial as transformações ao longo das *layers*. Como transformações, foi utilizado o Conv2D, sobre uma *layer* de *input* com o formato (256, 256, 3). O conv2D é uma *layer* convolucional que trabalha em duas dimensões, com os parâmetros de 16 filtros, para a primeira *layer*, um *kernel* de (3, 3) com um *stride* de 1. É utilizado a função de activação ReLu que retorna o valor do input se este for positivo e retorna o valor 0 se o contrário se verificar. Esta transformação introduz não-linearidade no modelo, tornando-o mais robusto.

Nas *layers* seguintes o número de filtros utilizados são, respectivamente, 32 e 64, sendo que as restantes transformações são equivalentes.

É realizado um MaxPooling2D entre cada *layer*, reduzindo as dimensões espaciais do mapa de atributos, e como consequência o número de atributos. Esta redução pode ajudar a evitar situações de *overfitting*, reduzir a complexidade computacional e tornar o modelo mais eficiente.

Após a realização do *pooling*, converte-se a *input data* de um vector multidimensional para um vector unidimensional, sem alterar o tamanho da *batch*, com o comando *Flatten*.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2D  (None, 127, 127, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 flatten (Flatten)           (None, 57600)             0

 dense (Dense)               (None, 256)               14745856

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 14,770,725
Trainable params: 14,770,725
Non-trainable params: 0
```

Figura 1 Estrutura do modelo

Finalmente, usando o comando *Dense*, aplica-se novamente a função ReLu numa instância e, numa seguinte, a função *softmax* que normaliza os valores dos atributos passando estes a estar compreendidos entre 0 e 1. Ao aplicarmos esta última função, permite-se ao modelo que atribua uma probabilidade elevada à classe correcta e uma probabilidade mais baixa a classes erradas, facilitando a previsão de uma imagem.

De forma a compilar o modelo pré treino, utilizou-se o optimizador “Adam” que adapta a *learning rate* individualmente para cada parâmetro, resultando numa melhor performance do mesmo. Utilizou-se também a *loss function* ‘SparseCategoricalCrossentropy’ que permite calcular a *cross-entropy loss* entre o resultado previsto e o resultado real. Este cálculo da perda permite aumentar o peso das previsões correctas e diminuir o peso dos atributos que promovem previsões erradas.

No treino foram realizados 20 *epochs* com conjuntos de 32 imagens, sendo validados contra o conjunto ‘val’ acima referido.

4. Resultados e Discussão

Os resultados do modelo apresentam-se nas Figuras 3, 4 e 5.

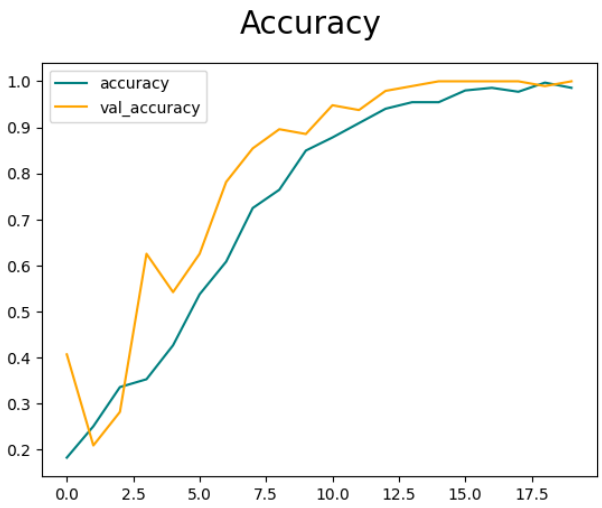

Figura 3 Evolução da precisão do modelo ao longo dos ephocs.

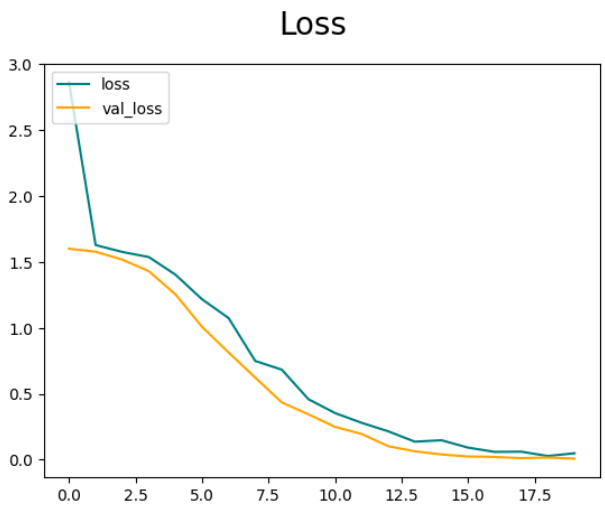

Figura 2 Evolução da loss do modelo ao longo dos ephocs.

Na Figura 4 e 5, observa-se que as curvas da *accuracy* e *loss* seguem uma progressão correcta. Embora haja alguns altos e baixos, estes não são indicativos de *overfitting* ou *underfitting*.

A matriz de confusão mostra a exactidão elevada obtida, revelando não haver confusão na classificação das castas por parte do modelo.

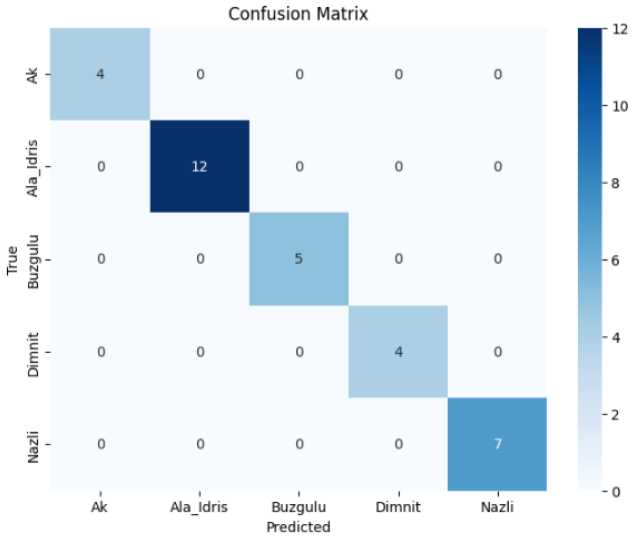

Figura 4 Matriz de confusão do modelo

5. Conclusão

Os resultados obtidos revelam a elevada exactidão e precisão do modelo desenvolvido no presente trabalho, com o objectivo de classificar imagens de 5 castas distintas de videira. Estes resultados permitem concluir que o modelo tem capacidade para classificar correctamente as imagens das folhas das diferentes castas.

O modelo poderia beneficiar de métodos de *data augmentation* por forma a torná-lo mais robusto e preciso. Embora tenha havido uma tentativa de implementar estas transformações no código os resultados não corresponderam às espectativas, possivelmente devido a um erro de código que não foi possível identificar.

Em trabalhos futuros, poderá ser interessante explorar a possibilidade de integrar métodos de *data augmentation* no modelo, assim como melhorar o código apresentado com a aplicação de métodos mais robustos ou mesmo através da utilização de diferentes bibliotecas da linguagem Python.

6. Contribuições

As elaborações do código e do relatório foram realizadas em conjunto pelos membros do trabalho.

7. Referências Bibliográficas

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., & Isard, M. (2016). TensorFlow: A System for Large-Scale Machine Learning. *OSDI*, 265–283. tensorflow.org

Gulli, A., & Pal, S. (2017). Deep learning with Keras. In *Packt Publishing Ltd.*

Harris, C. R., Millman, K. J., & van der Walt, S. J. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Renotte, N. (2022). *Build a deep cnn image classifier with any images*. https://www.youtube.com/watch?v=jztwpsIzEGc&t=2158s&ab_channel=NicholasRenotte