APVC – Desafio 1
Rede neuronal clássica para classificação de imagens de roupas

Descrição

Neste desafio pretende-se desenvolver uma rede neuronal clássica para classificar "mini-imagens" de roupa, calçado e malas. O *dataset* a utilizar é o FASHION_MNIST[1], também disponibilizado pelo tensorflow/keras[2]. Na Figura 1 podem-se observar alguns exemplos de imagens deste *dataset*.

Figura 1 - Exemplos de imagens do FASHION_MNIST

Para realização do trabalho sugere-se que use como base o código que poderá encontrar no script `fashionNet.py` fornecido. O script contém uma instrução que permite descarregar automaticamente o *dataset*, pelo que necessitará de uma ligação à internet.

Originalmente, o FASHION_MNIST consiste em 60.000 amostras para treino e 10.000 amostras para teste. O *dataset* original não contém um conjunto de validação e essa será uma questão a ter em conta no trabalho. Cada amostra é uma imagem em tons de cinzento com uma dimensão de 28x28 pixels. Existem 10 classes que representam os tipos de roupa/calçado/malas presentes no *dataset*:

Id classe	Significado	Id classe	Significado
0	*T-shirt/top*	**5**	*Sandal*
1	*Trouser*	**6**	*Shirt*
2	*Pullover*	**7**	*Sneaker*
3	*Dress*	**8**	*Bag*
4	*Coat*	**9**	*Ankle boot*

Embora os dados neste caso sejam imagens, pode-se aplicar uma rede neuronal clássica tratando o valor da luminância em cada pixel como se fosse um atributo (*feature*).

No entanto, como as imagens são matrizes bidimensionais, na definição da arquitetura da rede neuronal é necessário incluir uma camada adicional do tipo `Flatten`, que serve para transformar o *input* bidimensional num vetor, que poderá posteriormente ser tratado da mesma maneira que nos casos que foram vistos nas aulas. A inclusão da camada `Flatten` poderá ser feita de forma semelhante à do seguinte exemplo:

```
my_model = keras.Sequential([
                layers.Input(shape=(28, 28)),
                layers.Flatten(),

                # a partir daqui temos vetores e não matrizes 2D …
```

[1] https://github.com/zalandoresearch/fashion-mnist
[2] https://www.tensorflow.org/api_docs/python/tf/keras/datasets

Trabalho a desenvolver

Parte 1 – Implementar e avaliar uma rede neuronal para classificação multiclasse

Pretende-se implementar uma rede neuronal para classificação multiclasse[3].

a) Obtenha um conjunto de validação para monitorizar o treino da rede (divida o conjunto de treino original).

b) Construa um modelo de rede neuronal adequado, que seja capaz de classificar segundo as 10 classes do *dataset* (ver a secção anterior).

c) Compile a rede, escolhendo uma função de perda adequada e um algoritmo de otimização.

d) Treine o modelo – sugere-se que treine no máximo 50 épocas, e que use *callbacks* (ver o anexo) – espera-se que atinja uma taxa de acertos na ordem dos 85-90% no conjunto de teste.

e) Produza gráfico(s) que mostre(m) a evolução da perda e dos acertos durante o treino.

f) Calcule e mostre a taxa de acertos no conjunto de teste, após ter treinado o modelo.

g) Mostre uma matriz de confusão e determine e comente quais as classes que a rede confunde mais facilmente umas com as outras.

Parte 2 – Montar e avaliar uma rede neuronal para classificação binária

Nesta parte do trabalho pretende-se que desenvolva uma outra rede neuronal, usando o FASHION_MNIST, mas desta vez para realizar uma classificação binária segundo duas classes: *Vestuário* e *Calçado e Malas*.

Esta nova rede deverá ser implementada noutro *script* ou *notebook*.

a) Realize as adaptações necessárias ao *dataset* original de modo a ter as novas *labels*: 1 será a classe *Vestuário*, e 0 será a classe *Calçado e Malas* (ou vice-versa) – consulte a tabela da secção anterior para mapear as 10 classes originais nestas duas novas classes;
b) Adapte os passos b) a g) da Parte 1, aplicados agora a esta nova rede.

Parte 3 – Questão

Realize os cálculos necessários e compare de forma a conseguir dar uma resposta bem justificada à seguinte questão:

Para a classificação binária Vestuário/Calçado e Malas será melhor usar uma rede para classificação multiclasse e depois binarizar as predições da rede neuronal, ou será melhor usar uma rede neuronal projetada para realizar diretamente a classificação binária?

---

[3] Análoga ao exemplo dos fabricantes de vinho, apresentado nas aulas de 27 e 28/fevereiro

Entrega

A entrega do desafio é realizada através do *Moodle*. Basta que um dos elementos do grupo submeta o trabalho. Devem entregar os *scripts* ou os *notebooks* desenvolvidos. Deverá entregar também um pequeno documento pdf (max. 4 páginas) que mostre gráficos, resultados, comentários e explicações mais relevantes, ou então incluir essas informações nos próprios *notebooks* (caso entregue *notebooks*).

O *deadline* da entrega é o **dia 9/março (domingo) às 23:59**. Trabalhos entregues depois do *deadline* são aceites, mas penalizados com 1 ponto (em 10) por cada dia em atraso.

Avaliação

A nota será uma pontuação de 0 a 10 e será dado *feedback* através do *Moodle*. A avaliação de cada parte terá os seguintes pesos: Partes 1 e 2 – 40% cada; Parte 3 – 20%.

Em caso de dúvida esteja à vontade para contactar o professor.

Anexo – *Callbacks*

Os *callbacks* podem ser descritos como mecanismos utilitários que permitem a invocar funções automaticamente enquanto decorre o treino da rede neuronal (tipicamente invocam-se no final de cada época).

No script `fashionNet.py` estão definidos dois *callbacks*:

- `BEST_MODEL_CHECKPOINT` – permite gravar um ficheiro com os pesos do melhor modelo obtido até ao momento durante o treino, de acordo com um determinado critério. No *callback* que foi definido, o critério para ser o "melhor modelo" é ser o que obtém valor mais baixo na função de perda (ver os parâmetros `monitor` e `mode`). O ficheiro onde gravar é indicado no parâmetro `filepath` (que no script está regulado para ser `"tmp/best_model.weights.h5"`).

- `EARLY_STOPPING` – permite terminar o treino antecipadamente quando não se observem melhorias na função de perda durante um certo número de épocas consecutivas – esse número regula-se no parâmetro `patience`.

O dos *callbacks* é definido na função `fit(…)`, acrescendo um parâmetro `callbacks` com a lista dos *callbacks* que se quer utilizar:

```
history = my_model.fit(… , callbacks=[BEST_MODEL_CHECKPOINT, EARLY_STOPPING])
```

Depois da rede ser treinada podem-se carregar os pesos obtidos para o melhor modelo através da função `load_weights`, como no seguinte exemplo:

```
my_model.load_weights("tmp/best_model.weights.h5")
```

A partir desse ponto podem-se realizar as predições usando os melhores pesos obtidos durante todo processo de treino e não os que foram obtidos na última época, pois estes podem não ser necessariamente os melhores.

Poderá encontrar mais informação sobre os *callbacks* em:

https://blog.paperspace.com/tensorflow-callbacks/
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks