Trabalho Computacional 3

Rede Convolucional e Transfer Learning

1. Introdução e Base de Dados

Neste trabalho usaremos uma rede convolucional pré-treinada e a aplicaremos em um problema novo. Também experimentaremos com a divisão da base em treinamento, validação e teste, e usaremos validação para o "early stopping" na tentativa de controlar o sobre-ajuste. AProveitamos código publicado na internet por Gabriel Cassimiro.

A base de dados é a "TensorFlow Flowers Dataset". Ela contém 3670 imagens coloridas de flores pertencentes a uma de 5 classes: Margarida, Dente-de-leão, Rosa, Girassol e Tulipa.

Ela pode ser baixada com o código abaixo.

Obs: o módulo tensorflow_datasets a princípio não é acessível em instalações Windows. Você pode usar uma instalação local Unix, ou um serviço de núvem como o Google Colab ou Amazon Web Services.

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.utils import to_categorical
## Loading images and labels
(train_ds, train_labels), (test_ds, test_labels) = tfds.load(
    "tf_flowers",
    split=["train[:70%]", "train[:30%]"], ## Train test split
    batch_size=-1,
    as_supervised=True,  # Include labels
)

## Resizing images
train_ds = tf.image.resize(train_ds, (150, 150))
test_ds = tf.image.resize(test_ds, (150, 150))

## Transforming labels to correct format
train_labels = to_categorical(train_labels, num_classes=5)
test_labels = to_categorical(test_labels, num_classes=5)
print (train_ds.shape)
print (test_ds.shape)
```

```
(2569, 150, 150, 3)
(1101, 150, 150, 3)
```

Observe como foi feita a separação em 70% de dados para treinamento e 30% de dados para teste. Observe também que a base de dados original tem apenas um conjunto "train". A separação em treinamento, teste e validação é feita pelo usuário. Por isso a instrução para obter 70% do conjunto "train" e 30% do conjunto "train", o que soa estranho a princípio.

2. Treinando um MLP

Use esta base de dados para treinar um MLP, como feito no trabalho anterior com a base MNIST.

Escolha um MLP com 2 camadas escondidas. Não perca muito tempo variando a arquitetura porque este problema é difícil sem o uso de convoluções e o resultado não será totalmente satisfatório.

Você pode usar este código como base para definição da rede:

```
from tensorflow.keras import layers, models

flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(?, activation='relu')
dense_layer_2 = layers.Dense(?, activation='relu')
prediction_layer = layers.Dense(5, activation='softmax')


model = models.Sequential([
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
    prediction_layer
])
```

Observe que as imagens são achatadas (transformadas em vetor). Substitua as interrogações pelo tamanho desejado das camadas escondidas.

Neste problema vamos verificar o fenômeno do "over-fitting", e vamos tentar equilibrá-lo pela técnica de parada prematura de treinamento. Assim, dos dados de treinamento precisamos fazer uma nova separação para validação. Quando a acurácia de validação não sobe num dado número de épocas (o parâmetro "patience"), o treinamento é interrompido. Este trecho de código pode ser útil:

```
from tensorflow.keras.callbacks import EarlyStopping

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)


es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)

model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_size=32, callbacks=[es])
```

Os parâmetros dados são sugestões. Você agora pode testar o seu modelo, por exemplo, com:

```
# Evaluate the model on the test dataset
loss, accuracy = model.evaluate(test_ds, test_labels)

# Print the accuracy
print('Accuracy:', accuracy)
```

```
35/35 [==============================] - 177s 5s/step - loss: 0.2184 - accuracy: 0.9328
Accuracy: 0.9327883720397949
```

Mais uma vez, procure realizar ajustes, mas não espere um bom desempenho. Como dissemos, é um problema complexo de classificação de imagem, e é difícil fazer o MLP funcionar sozinho. Precisamos de um pré-processamento com base em uma rede convolucional.

3. Uso da rede VGG16 pré-treinada.

Lembre-se que a rede VGG usa como bloco básico cascata de convoluções com filtros 3x3, com "padding" para que a imagem não seja diminuída, seguida de um "max pooling" reduzindo imagens pela metade. O número de mapas vai aumentando e seu tamanho vai diminuindo ao longo de suas 16 camadas. Este é um modelo gigantesco e o treinamento com recursos computacionais modestos levaria dias ou semanas, se é que fosse possível.

No entanto, vamos aproveitar uma característica central das grandes redes convolucionais. Elas podem ser usadas como pré-processamento fixo das imagens, mesmo em um novo problema. (Lembre-se, a rede VGG original foi treinada na base ImageNet, que tem muitas categorias de imagem, não apenas flores).

O código abaixo realiza o download do modelo treinado, especifica que não será usada a última camada, e que os parâmetros do modelo-base não são ajustáveis.

Observe que a classe também tem o método preprocess_input para garantir que os dados sejam processados de maneira semelhante ao treinamento original da VGG16.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=train_ds[0].shape)
base_model.trainable = False ## Not trainable weights

## Preprocessing input
train_ds = preprocess_input(train_ds)
test_ds = preprocess_input(test_ds)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_no
58889256/58889256 [==============================] - 0s 0us/step
```

Você pode checar a arquitetura do modelo-base com o método "summary". Observe que há incríveis 14 milhões de parâmetros no modelo, que felizmente não vamos precisar adaptar.

```
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0
```

```
 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Agora, monte a rede final de forma semelhante ao MLP acima. Use o base_model como primeira camada, seguida de uma camada "flatten" (o MLP espera um vetor de entradas), e duas camadas densas. Elas não precisam ser muito grandes. Experimente com 50 e 20, respectivamente, ou algo próximo.

Volte a treinar e testar o modelo. Mesmo sem efetivamente treinar a rede VGG, ainda temos que passar os dados por ela a cada passo, e o treinamento é um tanto lento. Mas desta vez o problema deve ser resolvido satisfatoriamente.

4. Extras (opcionais)

4.1. Procure usar outra(s) redes convolucionais como base.

4.2. No lugar de "early stopping", experimente com regularização L1 e L2, e "dropout".

4.3. Procure (ou produza) imagens de algumas flores destas categorias, e teste em seu modelo.