Atividade Prática

Disciplina	**MAP Bootcamp**
Atividade	**Rede Neural – Extração de Características**

1. Objetivos

Neste exercício, você aprenderá como classificar imagens de cães e gatos usando o aprendizado de transferência de uma rede pré-treinada. Um modelo pré-treinado é uma rede salva que foi treinada anteriormente em um grande conjunto de dados, geralmente em uma tarefa de classificação de imagem em larga escala. Você usa o modelo pré-treinado como está ou usa o aprendizado de transferência para customizar esse modelo para uma determinada tarefa.

A intuição por trás do aprendizado de transferência para classificação de imagens é que, se um modelo for treinado em um conjunto de dados grande e geral o suficiente, esse modelo servirá efetivamente como um modelo genérico do mundo visual. Você pode aproveitar esses mapas de características aprendidas sem precisar começar do zero treinando um modelo grande em um grande conjunto de dados.

2. Enunciado

Você tentará duas maneiras de personalizar um modelo pré-treinado:

1. Extração de características: use as representações aprendidas por uma rede anterior para extrair características significativas de novas amostras. Você simplesmente adiciona um novo classificador, que será treinado a partir do zero, sobre o modelo pré-treinado, para que você possa adaptar novamente os mapas de características aprendidas anteriormente para o conjunto de dados. Você não precisa (re)treinar o modelo inteiro. A rede convolucional de base já contém características que são genericamente úteis para classificar imagens. No entanto, a parte final de classificação do modelo pré-treinado é específica para a tarefa de classificação original e subsequentemente específica para o conjunto de classes em que o modelo foi treinado.
2. Ajuste fino: descongele algumas das camadas superiores de uma base de

modelo congelada e treine em conjunto as camadas de classificação recém-adicionadas e as últimas camadas do modelo de base. Isso nos permite "ajustar" as representações de características de ordem superior no modelo base para torná-las mais relevantes para a tarefa específica.

Você seguirá o fluxo de trabalho geral de aprendizado de máquina.

1. Examine e entenda os dados.
2. Crie um pipeline de entrada, neste caso usando o Keras ImageDataGenerator.
3. Componha o modelo
    a. Carregar no modelo básico pré-treinado (e pesos pré-treinados).
    b. Empilhe as camadas de classificação na parte superior.
4. Treine o modelo.
5. Avalie modelo.

```
[ ]  import os

     import numpy as np

     import matplotlib.pyplot as plt
```

```
[1]  import tensorflow as tf
```

Processamento de Dados

Download dos dados

Use os conjuntos de dados TensorFlow para carregar o conjunto de dados de cães e gatos. Se quiser conferir as imagens, baixe o conjunto de imagens (768 Mb) de https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip ou de https://www.tensorflow.org/datasets/catalog/cats_vs_dogs.

Este pacote tfds é a maneira mais fácil de carregar dados predefinidos. Se você possui seus próprios dados e está interessado em importá-los com o TensorFlow, consulte o

carregamento de dados da imagem.

```
[ ]  import tensorflow_datasets as tfds
     tfds.disable_progress_bar()
```

O método tfds.load baixa e armazena em cache os dados e retorna um objeto tf.data.Dataset. Esses objetos fornecem métodos poderosos e eficientes para manipular dados e canalizá-los para o seu modelo.

Como "cats_vs_dogs" não define divisões padrão, use o recurso subsplit para dividi-lo em (treinamento, validação, teste) com 80%, 10% e 10% dos dados, respectivamente.

```
[ ]  (raw_train, raw_validation, raw_test), metadata = tfds.load(
         'cats_vs_dogs',
         split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
         with_info=True,
         as_supervised=True,
     )
```

Os objetos tf.data.Dataset resultantes contêm pares (imagem, rótulo - label) em que as imagens têm formato variável e 3 canais e o rótulo é escalar.

```
[ ]  print(raw_train)
     print(raw_validation)
     print(raw_test)
```

Mostre as duas primeiras imagens e rótulos do conjunto de treinamento:

```
[ ]  get_label_name = metadata.features['label'].int2str

     for image, label in raw_train.take(2):
       plt.figure()
       plt.imshow(image)
       plt.title(get_label_name(label))
```

Formate os dados

Use o módulo tf.image para formatar as imagens para a tarefa.

Redimensione as imagens para um tamanho de entrada fixo e redimensione os canais de entrada para um intervalo de [-1,1]

```
[ ]  IMG_SIZE = 160 # All images will be resized to 160x160

     def format_example(image, label):
       image = tf.cast(image, tf.float32)
       image = (image/127.5) - 1
       image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
       return image, label
```

Aplique essa função a cada item no conjunto de dados usando o método map:

```
[ ]  train = raw_train.map(format_example)
     validation = raw_validation.map(format_example)
     test = raw_test.map(format_example)
```

Agora embaralhe e agrupe os dados.

```
[ ]  BATCH_SIZE = 32
     SHUFFLE_BUFFER_SIZE = 1000
```

```
[ ]  train_batches = train.shuffle(SHUFFLE_BUFFER_SIZE).batch(BATCH_SIZE)
     validation_batches = validation.batch(BATCH_SIZE)
     test_batches = test.batch(BATCH_SIZE)
```

Inspecione um lote de dados:

```
[ ]  for image_batch, label_batch in train_batches.take(1):
        pass

     image_batch.shape
```

Crie o modelo base a partir dos convnets pré-treinados

Você criará o modelo base à partir do modelo **MobileNet V2** desenvolvido pelo Google. Ele foi pré-treinado no conjunto de dados ImageNet, um grande conjunto de dados que consiste em 1,4 milhões de imagens e 1000 classes. O ImageNet é um conjunto de dados de treinamento de pesquisa com uma ampla variedade de categorias, como jaca (fruta) e seringa (para injeções). Essa base de conhecimento nos ajudará a classificar cães e gatos de nosso conjunto de dados específico.

Primeiro, você precisa escolher qual camada do **MobileNet V2** usará para extração de características. A última camada de classificação (na parte superior, como a maioria dos diagramas dos modelos de aprendizado de máquina vai de baixo para cima) não é muito útil. Em vez disso, você seguirá a prática comum de depender da última camada antes da operação de nivelamento (flatten operation). Essa camada é chamada de "camada de gargalo". As características da camada mais ao fundo (bottleneck layer) retêm mais generalidade em comparação com a camada final / superior (top layer).

Primeiro, instancie um modelo **MobileNet V2** pré-carregado com pesos treinados no ImageNet. Ao especificar o argumento **include_top = False**, você carrega uma rede que não inclui as camadas de classificação na parte superior, o que é ideal para a extração de características.

```
[ ]  IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

     # Create the base model from the pre-trained model MobileNet V2
     base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                     include_top=False,
                                                     weights='imagenet')
```

Esse extrator de características converte cada imagem de 160x160x3 em um bloco de características de 5x5x1280. Veja o que ele faz no lote de imagens de exemplo:

```
[ ]  feature_batch = base_model(image_batch)
     print(feature_batch.shape)
```

Extração de características

Nesta etapa, você irá congelar a base convolucional criada a partir da etapa anterior e usar como extrator de característica. Além disso, você adiciona um classificador sobre ele e treina o classificador de nível superior (top-level).

Congelar a base convolucional

É importante congelar a base convolucional antes de compilar e treinar o modelo. O congelamento (configurando **layer.trainable = False**) impede que os pesos em uma determinada camada sejam atualizados durante o treinamento. O **MobileNet V2** possui muitas camadas, portanto, definir o flag treinável do modelo inteiro como False congelará todas as camadas (layers).

```
[ ]  base_model.trainable = False
```

```
[ ]  # Let's take a look at the base model architecture
     base_model.summary()
```

Para gerar previsões a partir do bloco de características, calcule a média sobre as localizações espaciais 5x5, usando uma camada tf.keras.layers.GlobalAveragePooling2D para converter as características em um único vetor de 1280 elementos por imagem.

```
[ ]  global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
     feature_batch_average = global_average_layer(feature_batch)
     print(feature_batch_average.shape)
```

Aplique uma camada tf.keras.layers.Dense para converter essas características em uma única previsão por imagem. Você não precisa de uma função de ativação aqui

porque esta previsão será tratada como um logit ou um valor bruto de previsão. Números positivos predizem a classe 1, números negativos predizem a classe 0.

```
[ ]  prediction_layer = tf.keras.layers.Dense(1)
     prediction_batch = prediction_layer(feature_batch_average)
     print(prediction_batch.shape)
```

Agora empilhe (stack) o extrator de características e essas duas camadas usando um modelo tf.keras.Sequential:

```
[ ]  model = tf.keras.Sequential([
       base_model,
       global_average_layer,
       prediction_layer
     ])
```

Compilar o modelo

Você deve compilar o modelo antes de treiná-lo. Como existem duas classes, use uma perda de entropia cruzada binária (binary cross-entropy loss) com from_logits = True, pois o modelo fornece uma saída linear.

```
[ ]  base_learning_rate = 0.0001
     model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate),
                   loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                   metrics=['accuracy'])
```

```
[ ]  model.summary()
```

Os parâmetros de 2,5 milhões no MobileNet estão congelados, mas existem 1,2 mil de parâmetros treináveis na camada Densa. Estes são divididos entre dois objetos variáveis, os pesos e os vieses.

```
[ ]  len(model.trainable_variables)
```

Treine o modelo

Após o treinamento por 10 epochs, você deverá ver ~ 96% de acurácia.

```
[ ]  initial_epochs = 10
     validation_steps=20

     loss0,accuracy0 = model.evaluate(validation_batches, steps = validation_steps)
```

```
[ ]  print("initial loss: {:.2f}".format(loss0))
     print("initial accuracy: {:.2f}".format(accuracy0))
```

```
[ ]  history = model.fit(train_batches,
                         epochs=initial_epochs,
                         validation_data=validation_batches)
```

Curvas de aprendizado da precisão (Learning Curves)

Vamos dar uma olhada nas curvas de aprendizado da acurácia / perda do treinamento e da validação ao usar o modelo base do **MobileNet V2** como um extrator de característica fixo.

```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']

     loss = history.history['loss']
     val_loss = history.history['val_loss']

     plt.figure(figsize=(8, 8))
     plt.subplot(2, 1, 1)
     plt.plot(acc, label='Training Accuracy')
     plt.plot(val_acc, label='Validation Accuracy')
     plt.legend(loc='lower right')
     plt.ylabel('Accuracy')
     plt.ylim([min(plt.ylim()),1])
     plt.title('Training and Validation Accuracy')

     plt.subplot(2, 1, 2)
     plt.plot(loss, label='Training Loss')
     plt.plot(val_loss, label='Validation Loss')
     plt.legend(loc='upper right')
     plt.ylabel('Cross Entropy')
     plt.ylim([0,1.0])
     plt.title('Training and Validation Loss')
     plt.xlabel('epoch')
     plt.show()
```

Nota: Se você está se perguntando por que as métricas de validação são claramente melhores que as métricas de treinamento, o principal fator é que camadas como tf.keras.layers.BatchNormalization e tf.keras.layers.Dropout afetam a precisão durante o treinamento. Eles são desativados ao calcular a perda de validação (validation loss).

Em um menor grau, é também porque as métricas de treinamento relatam a média de uma epoch, enquanto as métricas de validação são avaliadas após a epoch, portanto, as métricas de validação vêem um modelo que foi treinado um pouco mais.

Tuning fino

No experimento de extração de características, você treinava apenas algumas camadas sobre um modelo base do **MobileNet V2.** Os pesos da rede pré-treinada não foram atualizados durante o treinamento.

Uma maneira de aumentar ainda mais o desempenho é treinar (ou "ajustar") os pesos das camadas superiores do modelo pré-treinado, juntamente com o treinamento do

classificador adicionado. O processo de treinamento forçará os pesos a serem ajustados de mapas de características genéricas para características associadas especificamente ao conjunto de dados.

Nota: Isso só deve ser tentado depois de você treinar o classificador de nível superior com o modelo pré-treinado definido como não treinável. Se você adicionar um classificador inicializado aleatoriamente sobre um modelo pré-treinado e tentar treinar todas as camadas em conjunto, a magnitude das atualizações de gradiente será muito grande (devido aos pesos aleatórios do classificador) e seu modelo pré-treinado esquecerá o que aprendeu.

Além disso, você deve tentar ajustar um pequeno número de camadas superiores em vez de todo o modelo MobileNet. Na maioria das redes convolucionais, quanto mais alta a camada, mais especializada ela é. As primeiras camadas aprendem características muito simples e genéricas que generalizam para quase todos os tipos de imagens. À medida que você sobe, as características são cada vez mais específicas para o conjunto de dados no qual o modelo foi treinado. O objetivo do ajuste fino é adaptar essas características especializadas para trabalhar com o novo conjunto de dados, em vez de substituir o aprendizado genérico.

Descongele das camadas superiores do modelo

Tudo o que você precisa fazer é descongelar o base_model e definir as camadas inferiores para que não sejam treináveis. Em seguida, recompile o modelo (necessário para que essas alterações entrem em vigor) e reinicie o treinamento.

```
[ ]  base_model.trainable = True
```

```
[ ]  # Let's take a look to see how many layers are in the base model
     print("Number of layers in the base model: ", len(base_model.layers))

     # Fine-tune from this layer onwards
     fine_tune_at = 100

     # Freeze all the layers before the `fine_tune_at` layer
     for layer in base_model.layers[:fine_tune_at]:
       layer.trainable =  False
```

Compile o modelo

Compile o modelo usando uma taxa de aprendizado muito menor.

```
[ ]  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                   optimizer = tf.keras.optimizers.RMSprop(lr=base_learning_rate/10),
                   metrics=['accuracy'])
```

```
[ ]  model.summary()
```

```
[ ]  len(model.trainable_variables)
```

Se você treinou anteriormente para convergência, esta etapa melhorará sua precisão em alguns pontos percentuais.

```
[ ]  fine_tune_epochs = 10
     total_epochs =  initial_epochs + fine_tune_epochs

     history_fine = model.fit(train_batches,
                              epochs=total_epochs,
                              initial_epoch =  history.epoch[-1],
                              validation_data=validation_batches)
```

Vamos dar uma olhada nas curvas de aprendizado da acurácia / perda do treinamento e da validação ao ajustar as últimas camadas do modelo de base do **MobileNet V2** e treinar o classificador sobre ele. A perda de validação é muito maior do que a perda de

treinamento, portanto, você pode obter um ajuste excessivo (overfitting).

Você também pode sofrer uma adaptação excessiva, pois o novo conjunto de treinamento é relativamente pequeno e semelhante aos conjuntos de dados originais do **MobileNet V2**.

Após o ajuste fino, o modelo atinge quase 98% de precisão.

```
[ ]  acc += history_fine.history['accuracy']
     val_acc += history_fine.history['val_accuracy']

     loss += history_fine.history['loss']
     val_loss += history_fine.history['val_loss']
```

```
[ ]  plt.figure(figsize=(8, 8))
     plt.subplot(2, 1, 1)
     plt.plot(acc, label='Training Accuracy')
     plt.plot(val_acc, label='Validation Accuracy')
     plt.ylim([0.8, 1])
     plt.plot([initial_epochs-1,initial_epochs-1],
              plt.ylim(), label='Start Fine Tuning')
     plt.legend(loc='lower right')
     plt.title('Training and Validation Accuracy')

     plt.subplot(2, 1, 2)
     plt.plot(loss, label='Training Loss')
     plt.plot(val_loss, label='Validation Loss')
     plt.ylim([0, 1.0])
     plt.plot([initial_epochs-1,initial_epochs-1],
             plt.ylim(), label='Start Fine Tuning')
     plt.legend(loc='upper right')
     plt.title('Training and Validation Loss')
     plt.xlabel('epoch')
     plt.show()
```

6. Conclusão

Usando um modelo pré-treinado para extração de características: ao trabalhar com um pequeno conjunto de dados, é uma prática comum tirar proveito das características aprendidas por um modelo treinado em um conjunto de dados maior no mesmo domínio. Isso é feito instanciando o modelo pré-treinado e adicionando um classificador totalmente conectado na parte superior. O modelo pré-treinado é "congelado" e apenas os pesos do classificador são atualizados durante o treinamento. Nesse caso, a base convolucional extraiu todos as características associadas a cada imagem e você acabou de treinar um

classificador que determina a classe da imagem, considerando esse conjunto de características extraídas.

Ajuste fino de um modelo pré-treinado: para melhorar ainda mais o desempenho, é possível redirecionar as camadas de nível superior dos modelos pré-treinados para o novo conjunto de dados via ajuste fino. Nesse caso, você ajustou seus pesos para que seu modelo aprendesse características de alto nível específicos ao conjunto de dados. Essa técnica geralmente é recomendada quando o conjunto de dados de treinamento é grande e muito semelhante ao conjunto de dados original em que o modelo pré-treinado foi treinado.