Trabalho #6 - TensorBoard

Nesse trabalho você vai treinar uma rede convolucional para classificar imagens do conjunto de dados Fashion-MNIST e usar o TensorBoard para verificar como a matriz de confusão evolui ao longo do treinamento.

Coloque seu nome aqui

Nome: Bruno Rodrigues Silva

1. Importar bibliotecas e carregar o TensorBoard

Execute as células abaixo para importar as bilbiotecas necessárias e carregar o TensorBoard.

```
import io
import itertools
import numpy as np
import sklearn.metrics
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow import keras
from datetime import datetime

print("TensorFlow version: ", tf.__version__)
```

```
TensorFlow version:  2.3.0
```

```
# Carrega TensorBoard
%load_ext tensorboard
```

2. Carregar dados da Fashion-MNIST

O conjutno de dados Fashion-MNIST consiste de 70.000 imagens em tons de cinza de 10 tipos diferentes de vestuários, com 7.000 imagens por tipo. As imagens têm dimensão 28×28 pixels.

A primeira etapa é carregar as imagens. Esse conjunto de dados está disponível no Keras no formato de tensores NumPy e os dados já estão divididos nos conjuntos de treinamento e teste/validação.

Exercício #1: Carregar dados

Na célula abaixo inclua o seu código para carregar esse conjunto de dados. Mais detalhes de como carregar esse dados podem ser vistos em https://keras.io/api/datasets/fashion_mnist/#load_data-function

```
# Carregar base de dados
# Inclua seu código aqui (2 linhas)

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

# Dimensão dos dados
print('Dados de treinamento:', x_train.shape, y_train.shape)
print('Dados de validação:', x_test.shape, y_test.shape)
```

```
Dados de treinamento: (60000, 28, 28) (60000,)
Dados de validação: (10000, 28, 28) (10000,)
```

Saída esperada:

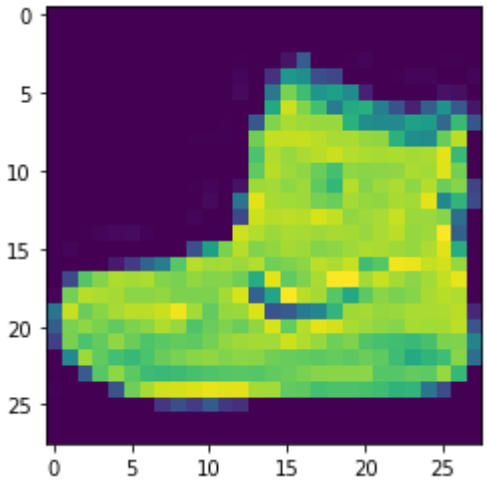
```
Dados de treinamento: (60000, 28, 28) (60000,)
Dados de validação: (10000, 28, 28) (10000,)
```

As classes das imagens são representadas por números inteiros, porém é mais interessante trabalhar com os nomes dos vestuários. Execute a célula abaixo para definir a lista `class-names` com os nomes das classes de vestuários. Cada nome da lista corresponde a um tipo de vestuário, por exemplo, 0 -> T-short/top, 1 -> Trouser, etc.

```
# Definição dos nomes das classes
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Mostra uma imagem de exemplo
index = 0
plt.imshow(x_train[index])
print("Número = " + str(class_names[y_train[index]]))
```

```
Número = Ankle boot
```



3. Formatação das imagens

O tensor de imagens de treinamento `x_train` tem dimensão `(60000, 28, 28)` e o de imagens de teste `x_test` tem de dimensão `(10000, 28, 28)`. Contudo, o TensorBoard espera receber tensores de imagens 4D, com os seguintes eixos `(batch_size, height, width, channels)`. Portanto, devemos redimensionar esses tensores para incluir o canal de cores. Na medida em que as imagens são em tons de cinza, devemos definir a dimensão `channels` como sendo `1`. Além disso, devemos normalizar as imagens para cada elemento ter um valor no intervalo `[0, 1]`.

Exercício #2: Redimensionamento e normalização das imagens

Na célula abaixo inclua o código necessário para realizar as operações de redimensionamento e normalização dos dados de entrada de treinamento e validação.

```
# Redimensiona imagens de treinamento e validação
# Inclua seu código aqui (2 linhas)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# Normalização das imagens de treinamento e validação
# Inclua seu código aqui (2 linhas)
x_train = x_train/255.
x_test  = x_test/255.
# Resultado
print('Dados de treinamento:', x_train.shape)
print('Dados de validação:', x_test.shape)
print('Verificação de alguns números:', x_train[10,10:20,10,0])
```

```
Dados de treinamento: (60000, 28, 28, 1)
Dados de validação: (10000, 28, 28, 1)
Verificação de alguns números: [0.65098039 0.62352941 0.61960784 0.59607843 0.576470
 0.63137255 0.65490196 0.65098039 0.62745098]
```

Saída esperada:

```
Dados de treinamento: (60000, 28, 28, 1)
Dados de validação: (10000, 28, 28, 1)
Verificação de alguns números: [0.65098039 0.62352941 0.61960784 0.59607843 0.57647059 0.60784314
 0.63137255 0.65490196 0.65098039 0.62745098]
```

4. Configuração e compilação da rede neural

Exercício #3: Configuração da RNA

Na célula abaixo crie uma RNA com a seguinte configuração:

- Duas camadas convolucionais de 64 filtros de dimensão 3x3 e função de ativação Relu;
- Após cada camada convolucional inclua uma camada de maxpooling com strid=2 e janela=2;
- Um camada densa dcom 128 neurônios e função de ativação Relu;
- Uma camda de saída com 10 neurônios (10 classes) e função de ativação softmax.

Não se esqueça da camada Flatten antes da primeira camada densa.

```
# Crie a RNA
# Inclua seu código aqui (~ 8 a 9 linhas)
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
rna = Sequential([
    layers.Conv2D(64, input_shape=(28, 28, 1), kernel_size=(3, 3), activation='relu'),
    layers.MaxPool2D(pool_size=(2,2), strides=2),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPool2D(pool_size=(2,2),strides=2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

rna.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        36928     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0         
_________________________________________________________________
dense (Dense)                (None, 128)               204928    
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290      
=================================================================
Total params: 243,786
Trainable params: 243,786
Non-trainable params: 0
_________________________________________________________________
```

Saída esperada:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 128)               204928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 243,786
Trainable params: 243,786
Non-trainable params: 0
_________________________________________________________________
```

5. Matriz de confusão

A matriz de confusão fornece informação detalhada sobre o desepenho de como RNA classificadora. Execute a célula abaixo para definir uma função que retorna uma figura criado com o Matplolib contendo a matriz de confusão.

```
def plot_confusion_matrix(cm, class_names):
    """
    Argumentos:
       cm (tensor, shape = [n, n]): matriz de confusão com classes representadas por nú
       class_names (tensor, shape = [n]): nomes das classes
    Retorna: figura matplotlib figure contendo a matriz de confusão
    """

    figure = plt.figure(figsize=(8, 8))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("Matriz de confusão")
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=45)
    plt.yticks(tick_marks, class_names)

    # Normaliza a matriz de confusão
    cm = np.around(cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)

```

```
    threshold = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        color = "white" if cm[i, j] > threshold else "black"
        plt.text(j, i, cm[i, j], horizontalalignment="center", color=color)

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    return figure
```

Exercício #4: Conversão da matriz de confusão em imagem PNG

Na célula abaixo crie uma função que recebe uma figura com a matriz de confusão, criada pela Matplotlib, e a converte em uma figura tipo PNG para ser visualizada no TensorBoard.

```
def plot_to_image(figure):
    """
    Converte figura matplotlib especificada por 'figure' em imagem tipo PNG. Após isso
    """

    # Salva figura na memória
    # Inclua seu código abaixo (2 linhas)
    mem_fig = io.BytesIO()
    plt.savefig(mem_fig, format='png')
    # Fecha figura para evitar erros
    # Inclua seu código abaixo (2 linhas)
    plt.close(figure)
    mem_fig.seek(0)
    # Converte PNG buffer em imagem do TensorFlow
    # Inclua seu código abaixo (1 linha)
    image = tf.image.decode_png(mem_fig.getvalue(), channels=4)
    # Adiciona eixo do batch_size
    # Inclua seu código abaixo (1 linha)
    image = tf.expand_dims(image, 0)

    return image
```

Exercício #5: Callbaks do TensorBoard

Complete a célula abaixo para criar um callback para o TensorBoard para visualizar a matriz de confusão e um gerador de dados para salvar a matriz de confusão. Se precisar de ajuda procure nas notas de aula e em [Keras TensorBoard callback].

```
# Apaga diretórios anteriores com a imagem da matriz de confusão
!rm -rf logs/image

# Cria um adiretório para salvar a matriz de confusão
logdir = "logs/image/" + datetime.now().strftime("%Y%m%d-%H%M%S")

# Define callback do TensorBoard para salvar custo, métrica e parâmetros
```

```
# Inclua seu código aqui (~1 linha)
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
file_writer_cm = tf.summary.create_file_writer(logdir + '/cm')
```

Exercício #6: Cálculo da matriz de confusão e callback para matriz de confusão

Na célula abaixo crie uma função que calcula uma matriz de confusão. Essa função deve realizar as seguintes operações:

- Calcula a matriz de confusão;
- Faz o gráfico da matriz de confusão com o MatPlotLib;
- Transforma a figura com a matriz de confusão em um figura tipo PNG;
- Com o gerador de dados `file_writer_cm` criado na célula anterior escreve os dados da matriz de confusão no diretório especificado.

No final essa função é usada para definir o `cm_callback` associado à geração da matriz de confusão para ser visualizada no TensorBoard.

```
def log_confusion_matrix(epoch, logs):
    """
    Essa função não retorna nada, ela somente salva no disco a
    """
    # Usa modelo para prever classes dos dados de validação
    # Inclua seu código aqui (~2 linhas)
    y_pred = rna.predict(x_test)
    y_pred_cl = np.argmax(y_pred, axis=1)


    # Calcula a matriz de confusão
    # Inclua seu código aqui (1 linha)
    cm = sklearn.metrics.confusion_matrix(y_test, y_pred_cl)
    # Faz o gráfico da matriz de confusão usando MatPlotLib
    # Inclua seu código aqui (~1 linha)
    figure = plot_confusion_matrix(cm, class_names=class_names)

    # Transforma matrix de confusão em imagem PNG
    # Inclua seu código aqui (~1 linha)
    cm_img = plot_to_image(figure)

    # Escreve matriz de confusão no diretório como uma "summary image".
    # Inclua seu código aqui (~ 2 linhas)
    with file_writer_cm.as_default():
        tf.summary.image("Confusion Matrix", cm_img, step=epoch)

# Define o callback per-epoch.
# Inclua seu código aqui (~1 linha)
cm_callback = tf.keras.callbacks.LambdaCallback(on_epoch_end=log_confusion_matrix)
```

Exercício #7: Treine a sua RNA e visualize os resultados no TensorBoard

Na célula abaixo crie um código para inicilizar o TensorBoard, compilar e treinar a sua RNA e depois visualizar os resultados no TensorBoard.

A compilação e treinamento da RNA deve ser feito usando os seguintes parâmetros:

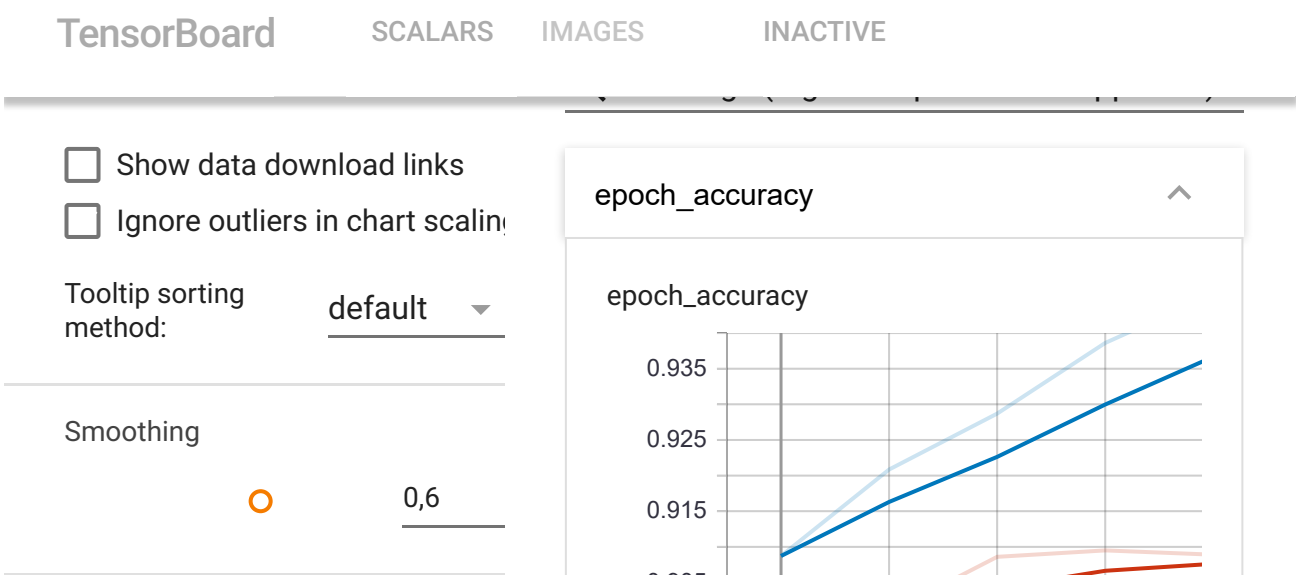
- Optimizador: Adam;
- Função de custo: `sparse_categorical_crossentropy`;
- Métrica: `accuracy`;
- Número de épocas: 5;
- Dados de validação: use `x_test` e `y_test`.

```
# Iniciliza o TensorBoard
# Inclua seu código aqui (1 linha)
%tensorboard --logdir logs/image

# Compilação da RNA
# Inclua seu código aqui (1 comando)
rna.compile('adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Treina a RNA com os dois callbacks
# Inclua seu código aqui (1 comando)
rna.fit(x_train, y_train, batch_size=32, epochs=5, callbacks=[tensorboard_callback, cm_
```

```
Reusing TensorBoard on port 6006 (pid 283), started 0:53:42 ago. (Use '!kill 283' to
Epoch 1/5
   1/1875 [..............................] - ETA: 0s - loss: 0.1482 - accuracy: 0.96
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2470 - accuracy:
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2103 - accuracy:
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1864 - accuracy:
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1627 - accuracy:
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1435 - accuracy:
<tensorflow.python.keras.callbacks.History at 0x7f59ce194f28>
```



Veja o resultado do treinamento no TENSORBOARD.

Saída esperada no seu notebook:

```
Reusing TensorBoard on port 6006 (pid 315), started 0:05:57 ago. (Use '!kill 315' to kill it.)

<IPython.core.display.Javascript object>

Epoch 1/5
1875/1875 - 90s - loss: 0.2173 - accuracy: 0.9188 - val_loss: 0.2578 - val_accuracy: 0.9043
Epoch 2/5
1875/1875 - 90s - loss: 0.1866 - accuracy: 0.9303 - val_loss: 0.2477 - val_accuracy: 0.9140
Epoch 3/5
1875/1875 - 90s - loss: 0.1663 - accuracy: 0.9369 - val_loss: 0.2668 - val_accuracy: 0.9088
Epoch 4/5
1875/1875 - 91s - loss: 0.1450 - accuracy: 0.9449 - val_loss: 0.2714 - val_accuracy: 0.9116
Epoch 5/5
1875/1875 - 91s - loss: 0.1283 - accuracy: 0.9515 - val_loss: 0.2880 - val_accuracy: 0.9097

<tensorflow.python.keras.callbacks.History at 0x7f73955626d8>
```

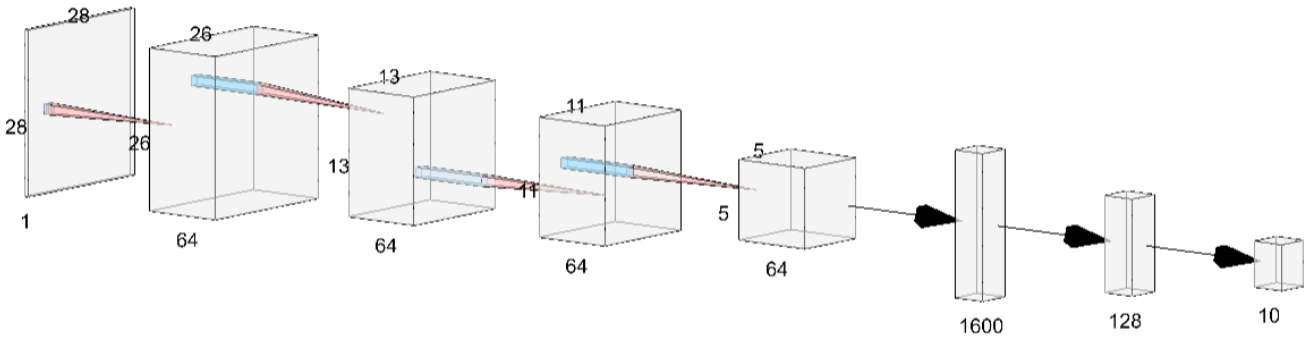
Execercíco #8: Conclusões:

Escreva as conclusões do seu trabalho nesse espaço.

A conclusão desse trabalho pode ser dividida em dois aspectos:

1. Análise dos resultados
2. Revisão do uso do Tensorboard para melhoria do processo de desenvolvimento

1. Análise dos resultados da Rede Neural recomendada



A estrutura sugerida para a rede neural segue diversas boas práticas na sua arquitetura e isso pode ser considerado como um indicador de que a rede terá um bom desempenho na tarefa de classificação multiclasse, - como demonstrado por [1] Simard em 2003 - algumas boas práticas são a utilização de camadas convolucionais para extração de um espaço de variáveis latentes sem um aumento na complexidade da implementação, uma vez que por conta destas não é necessária a aplicação de algoritmos mais complexos e mais específicos para que possa ser obtida uma exatidão alta; assim como vale ser mencionada a utilização da função de ativação ReLU ao oposto de funções não-lineares como a sigmoide ou a tanh, visto que a ReLU não sofre de alguns dos principais problemas encontrados com as outras mencionadas - como os problemas da dissipação e da explosão do gradiente descente -, além desta técnica, foram utilizadas duas camadas de MaxPooling ao invés de outras opções para redução da dimensionalidade - como camadas de AveragePooling - o que tende a fornecer um melhor resultado para a tarefa de classificação conforme [2] Andrade, 2019.
Após o treinamento da rede neural por 5 épocas, foi possível verificar que foi atingido um nível de exatidão próximo de 95% nos dados de treinamento e próximo de 91% nos dados de validação, o que pode ser considerado um desempenho bom, dado que não foi feito um processo aprofundado para a melhoria dos hiperparâmetros do modelo. Vale a pena mencionar que existem práticas que poderiam ter melhorado ainda mais o desempenho da rede, como o treinamento por mais épocas, a inclusão de dropout para evitar overfitting e a utilização de um gerador de imagens, visto que algumas das imagens podem ser trabalhadas com pré-processamento para um conjunto de treino mais significativo.

Referências Bibliográficas:
1 - Simard, Patrice & Steinkraus, Dave & Platt, John. (2003). Best Practices for Convolutional Neural Networks. Disponível em https://www.researchgate.net/publication/2880624_Best_Practices_for_Convolutional_Neural_Networks
2 - Andrade, Anderson. (2019). Best Practices for Convolutional Neural Networks Applied to Object Recognition in Images. disponível em https://arxiv.org/abs/1910.13029

2. Revisão do uso do Tensorboard para a melhoria no processo de desenvolvimento

O processo de desenvolvimento de uma rede neural para a resolução de um problema é uma atividade extremamente iterativa e em muitos casos experimental, uma vez que - em geral - buscamos resolver um problema de alta complexidade e que no caso de redes convolucionais não possui uma solução teórica que pode ser estimada antes do treinamento.
Baseado nisso é uma prática importante a manutenção de um espaço de testes de alta confiabilidade e que proporcione meios de customização, assim como é o Tensorboard, para que a evolução mencionada seja feita de maneira científica e reprodutível.