SCC0270 - Neural Networks and Deep Learning
Segundo Projeto Prático

Esse projeto tem como objetivo comparar um simples modelo de redes neurais convolucionais em bases de imagens com ruído. As bases de dados usadas nesse projeto são: CIFAR-10 e SVHN. A base de dados CIFAR-10 pode ser obtida no pacote **tensorflow.keras.datasets**, já a base SVHN está nos arquivos **train_32x32.mat** e **test_32x32.mat** (que foram compartilhados junto com esse enunciado).

Utilize a biblioteca Keras para criar os modelos pedidos. Você pode usar outras bibliotecas para fazer as demais análises =)
Para entender um pouco mais sobre o uso do Keras, veja este tutorial: https://www.tensorflow.org/datasets/keras_example

Todas as respostas devem ser justificadas com base em:

1. **Código Python mostrando a(s) análise(s) e/ou o(s) modelo(s) feitos;**
2. **O resultado da(s) análise(s) e/ou do(s) modelo(s) e**
3. **Uma explicação textual (pode ser breve) da conclusão obtida.**

Em caso de plágio (mesmo que parcial) o trabalho de todos os alunos envolvidos receberá nota ZERO.

NÃO enviar um link para o Google Colab como resposta/relatório do projeto.

Desorganização excessiva do código resultará em redução da nota do projeto.
Exemplos:

- **Códigos que devem ser rodados de forma não sequencial;**
- **Projeto entregue em vários arquivos sem um README;**
- **...**

Bom projeto,
Tiago.

Questão 1 (valor 2 pontos)

Você deve criar a seguinte CNN usando o Keras:

Input: 32x32x3
Conv2D: 16 neurônios (filters), kernels 3x3, ReLU
MaxPooling2D: pool 2x2
Conv2D: 16 neurônios (filters), kernels 3x3, ReLU
Conv2D: 16 neurônios (filters), kernels 3x3, ReLU
MaxPooling2D: pool 2x2
Conv2D: 32 neurônios (filters), kernels 3x3, ReLU
Conv2D: 32 neurônios (filters), kernels 3x3, ReLU
MaxPooling2D: pool 2x2
Dense: 10 neurônios

Dicas:

1. Use uma camada do tipo **Flatten** para transformar o output do terceiro max pooling (que será um tensor) em um vetor;
2. Use o parâmetro **padding='same'** em **TODAS** as camadas convolucionais, para que a convolução trate as bordas da imagem (e não haja redução no tamanho dos tensores).

Questão 2 (valor 2 pontos)

Carregue as duas bases de dados (CIFAR-10 e SVHN):

a) Ao carregar a base de dados CIFAR-10 você irá notar que y_train e y_test têm duas dimensões. Transforme eles em vetores.
b) Ao carregar a base de dados SVHN, e entender sua estrutura, você irá notar que X_train e X_test estão com as dimensões em uma ordem diferente da que usamos na aula (e que o Keras usa por padrão). Transforme tais arrays para ter a ordem que normalmente usamos (id da imagem, linhas, colunas e canais);
c) Transforme o valor dos pixels das imagens para o intervalo [0; 1];
d) Para cada uma das duas bases, compute:
    - Quantidade de imagens de treino e teste
    - Tamanho (número de linhas, colunas e canais) das imagens
    - Distribuição das classes

Dicas (que podem facilitar muito esse e os próximos itens):

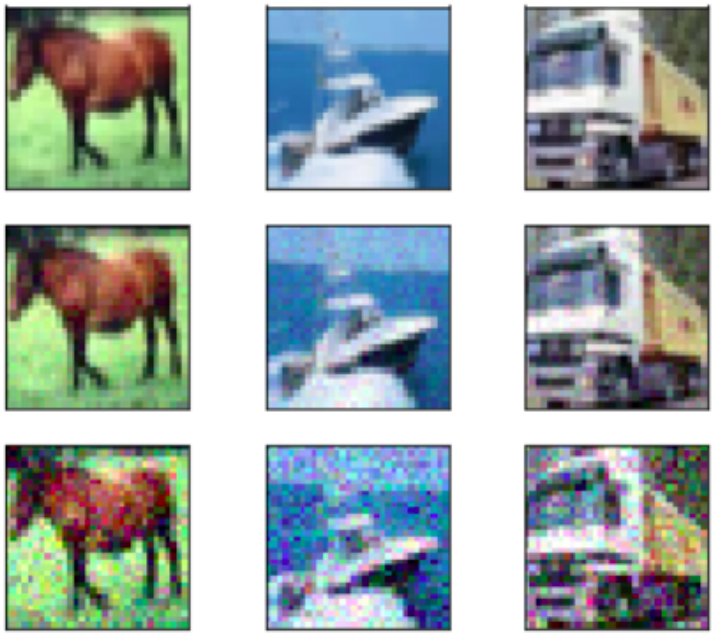
1. A função **loadmat** da biblioteca **scipy** permite fazer a leitura dos arquivos .mat (base SVHN);
2. Converter as imagens (X_train e X_test) para **float16** pode ajudar a reduzir o consumo de RAM. O numpy permite fazer tal conversão com um código do tipo: **np.float16(X)**;
3. Converter as classes (y_train e y_test) para uint8 permite reduzir o consumo de RAM.

Questão 3 (valor 2 pontos)

Vamos gerar versões ruidosas das bases de imagens e ver como isso afeta sua qualidade visual.

a) Gerar versões das duas bases de dados afetadas por ruído gaussiano com variâncias de 0.001 e 0.01, respectivamente;
b) Mostrar uma imagem de cada uma das classes em 3 versões (original, Gauss 0.001 e Gauss 0.01). Isso deve ser feito para as duas bases (CIFAR-10 e SVHN).

O ruído gaussiano gera imagens parecidas com as do seguinte exemplo, no qual temos: as imagens originais na primeira linha, imagens com ruído gaussiano (var 0.001) na segunda linha e imagens com ruído gaussiano (var 0.01) na terceira linha.



Dica:

1. Para gerar as imagens com ruído gaussiano use a função **random_noise** da biblioteca **scikit-image**. Tal função de ser usada da seguinte forma: **random_noise(img_original, mode='gaussian', var=0.01)**;

Questão 4 (valor 2 pontos)

Treinar uma versão do modelo para cada versão das bases de dados. Ou seja, você deve treinar os seguintes **seis** modelos:

1. Modelo treinado na CIFAR-10 original;
2. Modelo treinado na CIFAR-10 Gauss 0.001;
3. Modelo treinado na CIFAR-10 Gauss 0.01;
4. Modelo treinado na SVHN original;
5. Modelo treinado na SVHN Gauss 0.001;
6. Modelo treinado na SVHN Gauss 0.01;

Todos os modelos devem ser treinado usando:

- Adam como otimizador;
- Por 10 epocas;
- Os demais parâmetros devem ser deixados como padrão.

Dica:

- Você pode salvar os modelos e depois carregá-los novamente, com as funções **save** e **load_model**. Verifique a documentação do Keras.

Questão 5 (valor 2 pontos)

Agora, vamos tentar entender os impactos do ruído na acurácia dos modelos fazendo os seguintes experimentos:

a) Calcular a acurácia dos modelos em todas as versões da base de teste. Por exemplo, para o modelo treinado com os dados de treinamento da CIFAR-10 original devemos computar sua acurácia nas bases de teste original, Gauss 0.001 e Gauss 0.01. Ou seja, para cada um dos modelos vamos computar 3 acurácias;

b) Com bases nesses resultados, discuta qual das duas afirmações mais explica as variações de acurácia dos nossos experimentos:

   i) O ruído torna o problema mais difícil, logo fica mais difícil para o modelo aprender com dados ruidosos;

   ii) Quando um modelo é exposto a dados bastante diferentes dos de seu treinamento (e.g. com bastante ruído) sua acurácia pode cair.