Bruno Bacelar Abe 9292858

# T2 - Redes Neurais: Classificação usando Tensorflow com a base CIFAR-10

O trabalho foi feito usando uma placa de vídeo Nvidia 1050 ti (4GB), processador i5-6400, windows 10 x64, 8GB de ram e usando a versão 3.65 do Python.
Na rede, que está definida no arquivo *t2.py*, utilizamos a função de ativação relu (para as convoluções) e softmax (para a camada de saída). O otimizados escolhido foi o *Adadelta*, além de usarmos *crossentropy.*
A topologia da rede é a seguinte: 4 camadas convolucionais, 3 camadas de pooling e duas camadas densas, além de dropouts (para tentar evitar o *overfitting*).

1) Camadas convolucionais:
   a) Primeira: São 32 channels de saída, com janelas deslizantes de 5x5 e stride (1, 1) - que é o padrão - e utilizando a função de ativação relu;
   b) Segunda: São 64 channels de saída, com janelas deslizantes de 5x5 e stride (2, 2), utilizando a função de ativação relu;
   c) Outras: São 128 channels de saída, com janelas deslizantes de 3x3 e stride (1, 1), utilizando a função de ativação relu;
2) MaxPooling: São todas iguais, com tamanho de (2x2) e stride padrão (1,1);
3) Flatten: Transforma o resultado das camadas convolucionais em uma camada 1D, que pode ser tratada por uma rede densa (totalmente ligada);
4) Totalmente ligada: é constituída de duas camadas, uma com a função de ativação relu e a outra com a função de ativação softmax. A primeira é de tamanho 1000, enquanto a segunda possui tamanho 10 (que são o número de classes do modelo).

```
#Cria o modelo
model = keras.Sequential()
model.add(keras.layers.Conv2D(32, kernel_size=(5, 5), activation='relu', padding='same', input_shape=(32, 32, 3)))

model.add(keras.layers.Conv2D(64, kernel_size=(5, 5), activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
model.add(keras.layers.Dropout(0.25))

model.add(keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(keras.layers.Dropout(0.25))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1000, activation='relu'))
model.add(keras.layers.Dropout(0.5))

model.add(keras.layers.Dense(10, activation='softmax'))
```

(Figura 1: Topologia da rede)

Após o treinamento, que durou 300 épocas, é mostrado a acurácia esperada do modelo (no meu caso, por volta de 80% de acerto) e uma mensagem de sucesso.
Com o modelo criado, e treinado, escolhi algumas imagens (uma de cada classe) aleatoriamente na internet, para que fosse possível testar a rede. Para tratar as imagens, utilizo a biblioteca *OpenCV2* (cv2). Caso seja de interesse, na correção, é possível visualizar as imagens descomentando a função *showImages*.

```
38      #Mostra as imagens
39      #showImages(class_names, img)
```

(Figura 2: Mostrar imagens)

Após carregar as imagens pelo programa - *t2Read.py* - há o redimensionamento das imagens para um tamanho que se encaixe na rede (32 x 32) e também suas normatizações (divide-se cada imagem por 255 para normalizar o RGB) para então realizar a predição.


(Figura 3: Imagens)

A função de predição recebe um array com as imagens a serem testadas e retorna um vetor com as probabilidades da imagem em questão pertencer a cada uma das classes. Dessas probabilidades, escolhe-se a de maior valor e compara-se com o resultado esperado. Na base de teste, obtivemos uma acurácia de 80%, que era o esperado e compatível com o resultado do treino.

```
Quero:  airplane Obtido:  airplane
Quero:  automobile Obtido:  automobile
Quero:  bird Obtido:  bird
Quero:  cat Obtido:  cat
Quero:  deer Obtido:  deer
Quero:  dog Obtido:  dog
Quero:  frog Obtido:  bird
Quero:  horse Obtido:  airplane
Quero:  ship Obtido:  ship
Quero:  truck Obtido:  truck
```

(Figura 4: Resultados)

Em suma, os resultados obtidos com uma base de dados randomicamente escolhida batem com a acurácia retornada pelo treinamento da rede.
Para gerar o modelo da rede, basta executar o arquivo *t2.py*, após o término da execução, um arquivo chamado *my_model2.h5* será gerado. Para testar a base de imagens escolhidas, basta executar o programa *t2Read.py*.