MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo II - Redes Neurais Convolucionais (CNNs)

Exercícios com soluções

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Recomenda-se fortemente que os exercícios sejam feitos sem consultar as respostas antecipadamente.

Exercício 1)

Como se comparam uma camada densa e uma camada convolucional?

(a) Camadas densas são formadas por filtros que processam os dados localmente, enquanto as convolucionais permitem obter combinações lineares de todos os dados de entrada e por isso possuem menor quantidade de parâmetros
(b) As camadas convolucionais se diferenciam das densas por não assumir independência entre os dados de entrada
(c) As camadas densas realizam uma combinação linear de todos os elementos de entrada, enquanto as camadas convolucionais também processam todos os elementos da entrada, mas são consideradas não lineares
(d) Camadas convolucionais atuam sobre valores de campo receptivo local, processando localmente valores da entrada com os mesmos pesos, enquanto camadas densas possuem um peso para cada elemento da entrada.

Exercício 2)

A operação de convolução consiste em, centrada em um ponto de uma matriz ou tensor, fazer uma combinação linear de valores na região da posição central, gerando como saída um único valor, escalar.

Considere que a convolução opera numa região de tamanho 5×5 com stride = 1, isso significa que para cada conjunto local de 5×5 valores, produziremos um único de saída. Se a entrada é uma matriz com tamanho 10×10, e caso não seja

utilizado nenhum pré-processamento nos dados, qual o tamanho da saída?

(a) 10×10
(b) 6×6
(c) 8×8
(d) 5×5

Justificativa: para computar a saída em regiões de 5×5, precisamos começar na terceira linha e terceira coluna da matriz, e finalizar duas linhas e duas colunas antes da matriz terminar. Assim, para uma entrada de tamanho 10×10, perdemos duas linhas e duas colunas de cada lado, gerando uma matriz de 6×6 como saída.

Exercício 3)

Cada camada convolucional com k neurônios (filtros de convolução) gera k saídas distintas. Enquanto na camada densa as k saídas são valores individuais (escalares), na convolucional as saídas são mapas de ativação (ou mapas de características). Porque eles recebem esse nome?

(a) Pois podemos escolher os filtros de convolução de forma a processar imagens de entrada e obter características como borda, textura e regiões planas com cores.
(b) Pois podem ser interpretados como representações dos dados de entrada, aprendidas por meio da otimização da função de custo/perda da rede neural convolucional
(c) Pois os filtros convolucionais extraem de forma aleatória características da entrada de forma que essas características, ou ativações, melhorem a capacidade de classificar novos exemplos
(d) Pois foram baseadas em funções de ativação das redes neurais Multilayer Perceptron, as quais também realizam um tipo de processamento de características das entradas

---

Exercício 4)

Assuma uma entrada de tamanho $256 \times 256 \times 3$, e duas camadas convolucionais, sequenciais. A primeira possui 8 neurônios de tamanho $3 \times 3 \times 3$ e *não* usa zero-padding. A segunda possui 16 neurônios de tamanho $3 \times 3 \times p$ e aplica zero-padding.

Quantos mapas de ativação são gerados pela segunda camada convolucional, qual o tamanho final da saída da segunda camada e qual o valor de p?

(a) 16 mapas, saída $254 \times 254 \times 16$, $p = 8$
(a) 24 mapas, saída $254 \times 254 \times 8$, $p = 8$

(c) 16 mapas, saída $252 \times 252 \times 8$, $p = 16$
(d) 24 mapas, saída $256 \times 256 \times 16$, $p = 8$

Exercício 5)

Camadas convolucionais costumam gerar saídas de alta dimensionalidade, o que pode tornar difícil a otimização do modelo. Desejamos manter o aprendizado de mapas de ativação, mas reduzir a dimensionalidade espacial dos dados ao longo da rede neural. Marque a alternativa com a estratégia que melhor se adequa ao efeito desejado.

(a) Utilizar filtros de convolucão de tamanho menor
(b) Aumentar a quantidade de camadas de convolucão
(c) Utilizar camadas de pooling ou convoluções com stride maior do que 1
(d) Substituir todas as camadas convolucionais por camadas de pooling

---

Exercício 6)

Utilizando a biblioteca Keras, formule duas arquiteturas de rede neural sequencial convolucional, cuja entrada seja preparada para receber um vetor unidimensional de 1024 dimensões.

Arquitetura A

1. camada convolucional 1 com 16 filtros de tamanho 1×8, com padding
2. camada max pooling de tamanho 4
3. camada convolucional 2 com 32 filtros de tamanho 1×8, com padding
4. camada max pooling de tamanho 4
5. camada convolucional 3 com 8 filtros de tamanho 1×1, com padding
6. camada densa de saída com 3 neurônios.

Arquitetura B

1. camada convolucional 1 com 16 filtros de tamanho 1×8 e stride 4, com padding
2. camada convolucional 2 com 32 filtros de tamanho 1×8 e stride 4, com padding
3. camada convolucional 3 com 8 filtros de tamanho 1×1, com padding
4. camada densa de saída com 3 neurônios.

Qual o tamanho da saída das camadas convolucionais 2 e 3, e o total de parâmetros de cada rede?

(a) A: conv2=(256,32), conv3=(64,8), total=6075; B: conv2=(64,32), conv3=(64,8), total=4075

(b) A: conv2=(64,32), conv3=(64,8), total=6075; B: conv2=(64,32), conv3=(64,8), total=6075

(c) A: conv2=(64,32), conv3=(1,8), total=4128; B: conv2=(64,32), conv3=(1,8), total=1539

(d) A: conv2=(256,32), conv3=(64,8), total=6075; B: conv2=(64,32), conv3=(64,8), total=6075

In [1]:
```
import tensorflow as tf
from tensorflow import keras

modelA = keras.Sequential()
modelA.add(keras.layers.Conv1D(16, kernel_size=8, padding='same', ac
modelA.add(keras.layers.MaxPooling1D(pool_size=4))
modelA.add(keras.layers.Conv1D(32, kernel_size=8, padding='same', ac
modelA.add(keras.layers.MaxPooling1D(pool_size=4))
modelA.add(keras.layers.Conv1D(8, kernel_size=1, padding='same', act
modelA.add(keras.layers.Flatten())

modelA.add(keras.layers.Dense(3, activation='softmax'))
modelA.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 1024, 16)          144
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 256, 16)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 256, 32)           4128
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 64, 32)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 64, 8)             264
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 3)                 1539
=================================================================
Total params: 6,075
Trainable params: 6,075
Non-trainable params: 0
_________________________________________________________________
```

In [2]:
```
modelB = keras.Sequential()
modelB.add(keras.layers.Conv1D(16, kernel_size=8, strides=4, padding
modelB.add(keras.layers.Conv1D(32, kernel_size=8, strides=4, padding
modelB.add(keras.layers.Conv1D(8, kernel_size=1, padding='same', act
modelB.add(keras.layers.Flatten())
modelB.add(keras.layers.Dense(3, activation='softmax'))
modelB.summary()
```

```
Model: "sequential_1"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_3 (Conv1D)            (None, 256, 16)           144
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 64, 32)            4128
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 64, 8)             264
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539
=================================================================
Total params: 6,075
Trainable params: 6,075
Non-trainable params: 0
_________________________________________________________________
```

Exercício 7)

Utilizando a biblioteca Keras, formule uma arquiteturas de rede neural profunda, cuja entrada seja preparada para receber uma matriz com 1024×1024 dimensões e a seguinte arquitetura com 5 camadas:

1. camada convolucional 1 com 32 filtros de tamanho 3×3, com padding e stride 2 (nas duas direções)
2. camada convolucional 2 com 64 filtros de tamanho 1×3, com padding e stride $1, 2$
3. camada convolucional 3 com 64 filtros de tamanho 3×1, com padding e stride $2, 1$
4. camada convolucional 4 com 128 filtros de tamanho 3×3, sem padding e stride 3 (nas duas direções)
5. camada global average pooling

Qual o tamanho da saída das camadas conv2, conv3, conv4 e global_avg_pooling?

(a) conv2=(512,512,32), conv3=(256,256,64), conv4=(64,64,128) e global_avg_pooling=(128,128)
(b) conv2=(512,256,64), conv3=(256,256,64), conv4=(85,85,128) e global_avg_pooling=(128)
(c) conv2=(256,512,32), conv3=(256,256,64), conv4=(64,64,128) e global_avg_pooling=(128)
(d) conv2=(512,256,32), conv3=(128,256,64), conv4=(85,85,128) e global_avg_pooling=(128,128)

In [3]:
```
modelC = keras.Sequential()
modelC.add(keras.layers.Conv2D(32, kernel_size=(3,3), strides=(2,2),
modelC.add(keras.layers.Conv2D(64, kernel_size=(1,3), strides=(1,2),
modelC.add(keras.layers.Conv2D(64, kernel_size=(3,1), strides=(2,1),
modelC.add(keras.layers.Conv2D(128, kernel_size=(3,3), strides=(3,3)
modelC.add(keras.layers.GlobalAveragePooling2D())
modelC.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 512, 512, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 512, 256, 64)      6208
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 256, 256, 64)      12352
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 85, 85, 128)       73856
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
=================================================================
Total params: 92,736
Trainable params: 92,736
Non-trainable params: 0
_________________________________________________________________
```

Exercício 8)

Carregue a base de dados CIFAR-100 e exiba as 10 primeiras imagens dessa base de dados. Normalize os dados das imagens de forma a que os valores estejam entre 0 e 1, depois converta as classes para o tipo categórico utilizando o `tf.keras.utils.to_categorical` .

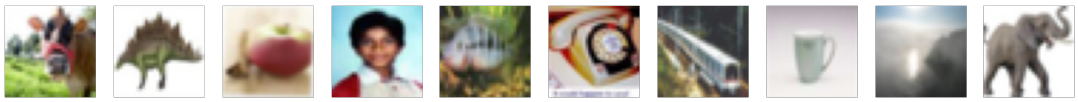
A seguir, adapte a arquitetura da questão anterior para que a rede seja capaz de receber como entrada imagens dessa base. Crie uma camada densa no final que permita classificar as imagens nos rótulos disponíveis no conjunto de treinamento. Todas as camadas da rede, exceto a última, devem ter funções de ativação do tipo `relu` .

Qual o tamanho da saída das duas últimas camadas da rede (global average pooling e densa), e o total de parâmetros da mesma rede?

(a) global average pooling = 128, densa = 128, parâmetros = 106212
(b) global average pooling = 128, densa = 100, parâmetros = 12900
(c) global average pooling = 128, densa = 20, parâmetros = 12900
(d) global average pooling = 128, densa = 100, parâmetros = 106212

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar100
(x_train, y_train), (x_test, y_test) = cifar100.load_data()

n_imgs = 10
plt.figure(figsize=(16,2))
for im in np.arange(n_imgs):
    plt.subplot(1,n_imgs,im+1)
    plt.imshow(x_train[im])
    plt.axis('off')

img_lin, img_col, img_cha = x_train.shape[1], x_train.shape[2], x_tr
num_classes = len(np.unique(y_train))
print(x_train.shape)
print('Classes: ', num_classes)

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

```
(50000, 32, 32, 3)
Classes:  100
```



In [5]:
```
modelD = keras.Sequential()
modelD.add(keras.layers.Conv2D(32, kernel_size=(3,3), strides=(2,2),
modelD.add(keras.layers.Conv2D(64, kernel_size=(1,3), strides=(1,2),
modelD.add(keras.layers.Conv2D(64, kernel_size=(3,1), strides=(2,1),
modelD.add(keras.layers.Conv2D(128, kernel_size=(3,3), strides=(3,3)
modelD.add(keras.layers.GlobalAveragePooling2D())
modelD.add(keras.layers.Dense(num_classes, activation="softmax"))
modelD.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 16, 16, 32)        896
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 8, 64)         6208
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 64)          12352
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 2, 2, 128)         73856
_________________________________________________________________
global_average_pooling2d_1 ( (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               12900
=================================================================
Total params: 106,212
```

```
Trainable params: 106,212
Non-trainable params: 0
```

Exercício 9)

Defina as sementes aleatórias do numpy para 1 e do tensorflow para 2. Depois, utilizando a arquitetura definida no exercício anterior, configure a rede para treinar com a configuração abaixo, salvando o histórico da perda e acurácia para as épocas.

- otimizador: SGD
- taxa de aprendizado: 0.05
- função de custo: `categorical_crossentropy`
- métrica: `accuracy`
- épocas: 8
- batchsize: 100

Após as 8 épocas, exiba e analise gráfico da função de custo no conjunto de treinamento ao longo das épocas e avalie a acurácia no conjunto de testes. Considerando: (1) a convergência da função de custo e (2) comparando a acurácia no conjunto de testes com o que seria a acurácia de um classificador aleatório para a base de dados CIFAR-100, escolha a alternativa correta.

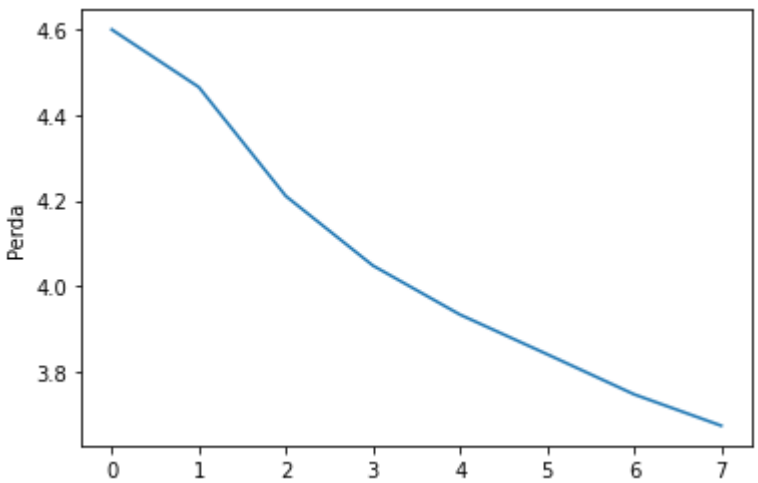
(a) A função de custo indica um valor baixo de custo e assim, treinar por mais épocas não irá beneficiar o modelo, a acurácia obtida é mais do que 100 vezes superior do que um classificador aleatório
(b) A função de custo indica um valor baixo de custo e assim, treinar por mais épocas não irá beneficiar o modelo, a acurácia obtida é apenas ligeiramente superior a um classificador aleatório
(c) A função de custo indica estabilização, alcançando uma acurácia 20 vezes superior a de um classificador aleatório
(d) A função de custo indica que ainda há espaço para o treinamento de mais épocas, e a acurácia obtida é mais do que 10 vezes superior do que um classificador aleatório

In [6]:
```
from numpy.random import seed
seed(1)
from tensorflow.random import set_seed
set_seed(2)

modelD.compile(loss='categorical_crossentropy',
               optimizer=tf.keras.optimizers.SGD(lr=0.05),
               metrics=['accuracy'])
```

In [7]:
```
batch_size = 100
epochs = 8

# a variável history guarda os dados do processo de treinamento para
# posteriormente analisarmos
history = modelD.fit(x_train, y_train,
                     batch_size=batch_size,
                     epochs=epochs,
                     verbose=1)
```

```
Epoch 1/8
500/500 [==============================] - 11s 21ms/step - loss: 4.5
997 - accuracy: 0.0097
Epoch 2/8
500/500 [==============================] - 13s 25ms/step - loss: 4.4
651 - accuracy: 0.0262
Epoch 3/8
500/500 [==============================] - 15s 31ms/step - loss: 4.2
102 - accuracy: 0.0525
Epoch 4/8
500/500 [==============================] - 17s 34ms/step - loss: 4.0
477 - accuracy: 0.0759
Epoch 5/8
500/500 [==============================] - 19s 37ms/step - loss: 3.9
331 - accuracy: 0.0976
Epoch 6/8
500/500 [==============================] - 20s 40ms/step - loss: 3.8
405 - accuracy: 0.1143
Epoch 7/8
500/500 [==============================] - 19s 38ms/step - loss: 3.7
466 - accuracy: 0.1306
Epoch 8/8
500/500 [==============================] - 20s 39ms/step - loss: 3.6
734 - accuracy: 0.1433
```

In [8]:
```
plt.plot(history.history['loss'])
plt.ylabel('Perda')
plt.show()
```



In [9]:
```
score = modelD.evaluate(x_test, y_test, verbose = 0)
print("Acurácia: %.4f (aleatório = %.4f)" % (score[1], 1/num_classes
```

```
Acurácia: 0.1357 (aleatório = 0.0100)
```

Exercício 10)

Crie uma nova rede neural com base na anterior mas adicionando, logo após a 3.a camada convolucional, e antes da 4.a camada convolucional, outras 2 camadas no formato:

- camada convolucional com 64 filtros de tamanho 1×3, com padding e stride $1, 2$
- camada convolucional com 64 filtros de tamanho 3×1, com padding e stride $2, 1$

Defina as sementes aleatórias do numpy para 1 e do tensorflow para 2, e a seguir execute o treinamento dessa rede utilizando os parâmetros conforme a questão anterior, mas aumente o número de épocas para 16.

Considerando:

1. a diferença no valor da função de custo obtida nas épocas 1 e 8, e
2. a acurácia no conjunto de testes após o treinamento completo

Escolha a alternativa correta:

(a) A diferença do custo da 1.a para a 8.a época é maior nessa nova arquitetura, indicando convergência mais rápida, e ao final a acurácia atinge valor superior a 25% com a arquitetura mais profunda
(b) A diferença do valor da função de custo da 1.a para a 8.a época é menor nessa nova arquitetura, indicando convergência mais lenta, e ao final a acurácia atinge valor superior a 25% com essa arquitetura mais profunda
(c) A diferença do valor da função de custo da 1.a para a 8.a época é maior nessa nova arquitetura, mostrando a superioridade da arquitetura mais profunda, que ao final atinge acurácia atinge valor acima de 50%
(d) A diferença do valor da função de custo da 1.a para a 8.a época é menor nessa nova arquitetura, indicando que a arquitetura menos profunda é mais estável, e ao final a acurácia atinge valor próxmo a 10% com essa arquitetura mais profunda

In [10]:
```
modelE = keras.Sequential()
modelE.add(keras.layers.Conv2D(32, kernel_size=(3,3), strides=(2,2),
modelE.add(keras.layers.Conv2D(64, kernel_size=(1,3), strides=(1,2),
modelE.add(keras.layers.Conv2D(64, kernel_size=(3,1), strides=(2,1),
modelE.add(keras.layers.Conv2D(64, kernel_size=(1,3), strides=(1,2),
modelE.add(keras.layers.Conv2D(64, kernel_size=(3,1), strides=(2,1),
modelE.add(keras.layers.Conv2D(128, kernel_size=(3,3), padding='vali
modelE.add(keras.layers.GlobalAveragePooling2D())
modelE.add(keras.layers.Dense(num_classes, activation="softmax"))
modelE.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 16, 16, 32)        896
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 8, 64)         6208
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 64)          12352
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 4, 64)          12352
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 4, 4, 64)          12352
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 2, 2, 128)         73856
_________________________________________________________________
global_average_pooling2d_2 ( (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               12900
=================================================================
Total params: 130,916
Trainable params: 130,916
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
seed(1)
set_seed(2)

modelE.compile(loss='categorical_crossentropy',
              optimizer=tf.keras.optimizers.SGD(lr=0.05),
              metrics=['accuracy'])

batch_size = 100
epochs = 16

# a variável history guarda os dados do processo de treinamento para
# posteriormente analisarmos
history2 = modelE.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

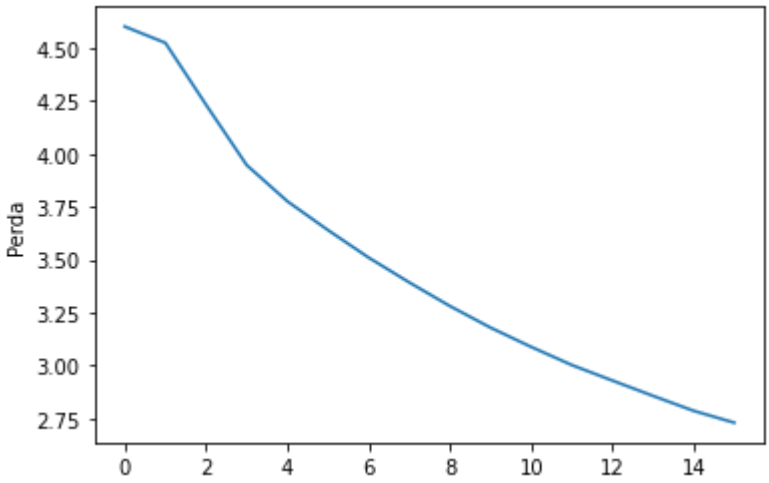
```
Epoch 1/16
500/500 [==============================] - 27s 53ms/step - loss: 4.6
```

```
027 - accuracy: 0.0089 - val_loss: 4.5959 - val_accuracy: 0.0146
Epoch 2/16
500/500 [==============================] - 31s 62ms/step - loss: 4.5
255 - accuracy: 0.0210 - val_loss: 4.3831 - val_accuracy: 0.0349
Epoch 3/16
500/500 [==============================] - 29s 58ms/step - loss: 4.2
331 - accuracy: 0.0542 - val_loss: 4.0212 - val_accuracy: 0.0772
Epoch 4/16
500/500 [==============================] - 28s 56ms/step - loss: 3.9
477 - accuracy: 0.0932 - val_loss: 3.8346 - val_accuracy: 0.1114
Epoch 5/16
500/500 [==============================] - 28s 57ms/step - loss: 3.7
766 - accuracy: 0.1241 - val_loss: 3.7016 - val_accuracy: 0.1342
Epoch 6/16
500/500 [==============================] - 34s 68ms/step - loss: 3.6
402 - accuracy: 0.1463 - val_loss: 3.5683 - val_accuracy: 0.1624
Epoch 7/16
500/500 [==============================] - 35s 70ms/step - loss: 3.5
103 - accuracy: 0.1695 - val_loss: 3.4960 - val_accuracy: 0.1711
Epoch 8/16
500/500 [==============================] - 38s 76ms/step - loss: 3.3
928 - accuracy: 0.1903 - val_loss: 3.3881 - val_accuracy: 0.1913
Epoch 9/16
500/500 [==============================] - 34s 68ms/step - loss: 3.2
811 - accuracy: 0.2110 - val_loss: 3.3046 - val_accuracy: 0.2120
Epoch 10/16
500/500 [==============================] - 35s 70ms/step - loss: 3.1
790 - accuracy: 0.2294 - val_loss: 3.2023 - val_accuracy: 0.2277
Epoch 11/16
500/500 [==============================] - 31s 62ms/step - loss: 3.0
878 - accuracy: 0.2478 - val_loss: 3.1067 - val_accuracy: 0.2521
Epoch 12/16
500/500 [==============================] - 31s 63ms/step - loss: 3.0
022 - accuracy: 0.2644 - val_loss: 3.1205 - val_accuracy: 0.2411
Epoch 13/16
500/500 [==============================] - 31s 61ms/step - loss: 2.9
287 - accuracy: 0.2768 - val_loss: 2.9962 - val_accuracy: 0.2693
Epoch 14/16
500/500 [==============================] - 32s 63ms/step - loss: 2.8
562 - accuracy: 0.2914 - val_loss: 2.9884 - val_accuracy: 0.2717
Epoch 15/16
500/500 [==============================] - 35s 70ms/step - loss: 2.7
856 - accuracy: 0.3052 - val_loss: 2.9495 - val_accuracy: 0.2819
Epoch 16/16
500/500 [==============================] - 37s 74ms/step - loss: 2.7
294 - accuracy: 0.3182 - val_loss: 2.9449 - val_accuracy: 0.2801
```

In [12]:
```
score = modelE.evaluate(x_test, y_test, verbose = 0)
print("Acurácia: %.4f (aleatório = %.4f)" % (score[1], 1/num_classes
```

```
Acurácia: 0.2801 (aleatório = 0.0100)
```

In [13]:
```
plt.plot(history2.history['loss'])
plt.ylabel('Perda')
plt.show()
```


Perda
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
0
2
4
6
8
10
12
14