MBA em Ciência de Dados

Redes Neurais e Arquiteturas Profundas

Módulo IV - Estratégias de Treinamento e Transferência de Aprendizado

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)

Em termos do viés do algoritmo de aprendizado, o qual está relacionado ao espaço de funções admissíveis a partir do qual esse algoritmo gera modelos (de classificação, regressão, etc.), como podemos comparar métodos rasos, comumente com poucos parâmetros a ajustar, e profundos, comparativamente possuindo mais parâmetros?

(a) Métodos profundos tem viés mais fraco quando comparado ao métodos rasos
(b) Métodos profundos tem viés mais forte quando comparado ao métodos rasos
(c) Métodos profundos possuem viés enquanto os rasos não possuem viés
(d) Métodos profundos não possuem viés enquanto os rasos possuem viés

Justificativa: Métodos profundos possuem mais parâmetros do que métodos rasos e portanto podem aproximar maior quantidade de funções admissíveis, isso faz com que seu viés seja fraco em comparação a métodos rasos

---

Exercício 2)

Modelos de redes neurais com alta capacidade, isso é, com grande quantidade de parâmetros, são capazes de:

(a) Generalizar muito bem sempre, para qualquer tipo de dados futuros
(b) Gerar underfitting pois podem não conseguir convergir e portanto não se ajustam aos dados de treinamento
(c) Memorizar o conjunto de treinamento, ajustando até mesmo rótulos aleatórios
(d) Obter alta robustez com relação à possíveis ataques

Justificativa: se possuirem capacidade suficiente, podem memorizar o conjunto de treinamento completo com rótulos aleatórios. Modelos com muitos parâmetros podem falhar em generalizar, comumente não geram underfitting - pelo contrário, tendem a obter overfitting nos dados de treinamento, e não há garantia de robustez à ataques.

Exercício 3)

Métodos diretamente relacionados a evitar que redes neurais profundas convirjam para o modelo "memorizador" incluem:

(a) Evitar treinar por número de épocas excessivas, técnicas de regularização e obtenção de conjuntos de treinamento maiores
(b) Regularização, transferência de aprendizado e emprego de maior número de neurônios por camada
(c) Ajustar corretamente o tamanho do batch, utilizar dropout e evitar o uso da função de ativação softmax
(d) Regularização, aumento da quantidade de dados de treinamento e uso de funções de ativação do tipo ReLU

Justificativa: as técnicas: transferência de aprendizado, ajuste do tamanho do batch, aumento de quantidade de neurônios por camada, função de ativação softmax ou relu não estão diretamente relacionadas ao efeito de memorização ou overfitting de modelos.

Exercício 4)

Considere uma base de dados de imagens para treinamento, com 3 classes, e 20 exemplos por classe. Suponha que deseje utilizar uma CNN como solução para obter um classificador para esse problema. Qual das opções abaixo é a mais viável?

(a) Carregar um modelo de CNN bem estabelecido, pré-treinado em base de dados grande, obter os mapas de ativação de alguma camada da CNN e utilizar esses mapas como entrada para um classificador com maior garantia de aprendizado, como o SVM
(b) Carregar um modelo de CNN bem estabelecido, com pesos inicializados aleatoriamente, obter os mapas de ativação de alguma camada da CNN e utilizar esses mapas como entrada para um classificador com maior garantia de aprendizado, como o SVM
(c) Carregar um modelo de CNN bem estabelecido, com pesos inicializados aleatoriamente, projetar uma nova camada de saída densa com 3 neurônios, e treinar apenas essa camada de saída como classificador
(d) Carregar um modelo de CNN bem estabelecido, pré-treinado em base de dados grande, projetar uma nova camada de saída densa com 3 neurônios, e treinar toda a CNN com os dados de treinamento, incluindo a nova camada

Justificativa: Com apenas 20 exemplos por classe é inviável treinar uma CNN a partir de pesos aleatórios. Ainda que seja possível carregar pesos pré-treinados, treinar os pesos de toda uma CNN conforme a alternativa (d) indica seria ainda arriscado. Assim, a solução mais viável é utilizar os mapas de ativação como características para classificadores externos.

Exercício 5)

Normalização dos dados entre camadas de uma rede profunda tem a função principal de:

(a) Permitir melhor visualização dos mapas de ativação
(b) Evitar underfitting gerando magnitudes de gradientes mais extremas
(c) Facilitar a convergência suavizando a descida do gradiente e reduzindo o problema de desaparecimento do gradiente
(d) Evitar overfitting padronizando os dados do batch aleatoriamente

Justificativa: a normalização auxilia na convergência ou treinamento do modelo, em particular suavizando a descida do gradiente e reduzindo o problema de desaparecimento do gradiente. A normalização não gera magnitudes de gradiente mais extremas, nem padroniza os dados de forma aleatória.

Exercício 6)

Carregue a base de dados Boston Housing, e padronize os dados (conforme código abaixo).

A seguir, considere as seguintes arquitetura de rede neural com camadas densas, todas com **6** neurônios nas camadas intenas/ocultas:

Arquitetura A (12 camadas):

- 12 camadas densas com ativação `relu`
- 1 camada densa com 1 neurônio e ativação `relu`

Arquitetura B (12 camadas com normalização em batch):

- 2 camadas densas com ativação `relu`
- 1 camada de normalização em batch
- 2 camadas densas com ativação `relu`
- 1 camada de normalização em batch
- 2 camadas densas com ativação `relu`
- 1 camada de normalização em batch
- 2 camadas densas com ativação `relu`
- 1 camada de normalização em batch
- 2 camadas densas com ativação `relu`
- 1 camada de normalização em batch
- 2 camadas densas com ativação `relu`
- 1 camada densa com 1 neurônio e ativação `relu`

Faremos 3 experimentos de treinamento, a partir de 3 configurações de sementes distintas. Dica: monte cada arquitetura em uma função do Python e retorne o modelo para facilitar o processo.

Você deverá inicializar as sementes antes de criar cada modelo (A e B) na memória. Logo a seguir, compilar e treinar, passando o conjunto de teste como "validação" para verificarmos possíveis problemas de generalização.

1. Experimento 1: seed(1) e set_seed(1)
2. Experimento 2: seed(2) e set_seed(2)
3. Experimento 3: seed(3) e set_seed(3)

Em cada experimento, treine as redes A e B com a função de custo `mse` , o otimizador Adam com taxa de aprendizado 0.01, 50 épocas, e tamanho de batch 32. Salve os históricos por época e trace o gráfico da função de custo computada no treinamento e no teste ao longo das épocas dos modelos A e B para cada um dos experimentos.

Observando os gráficos dos 3 experimentos, podemos dizer sobre o processo de convergência do valor de custo que:

(a) A: convergiu sempre atingindo bons valores de erro médio quadrático; B: convergiu rapidamente sempre, porém com overfitting do modelo em todos os casos
(b) A: não convergiu nos experimentos; B: não convergiu nos experimentos
(c) A: convergiu rapidamente em todos os experimentos, mas o modelo não melhorou mais o erro após as primeiras épocas; B: não convergiu na maioria dos casos, mas quando convergiu apresentou melhor generalização do que A
(d) A: convergiu para um valor baixo de erro, mas não em todos os casos; B: convergiu em todos os casos, ainda que o erro na validação apresente irregularidades

Justificativa: Ver código abaixo.

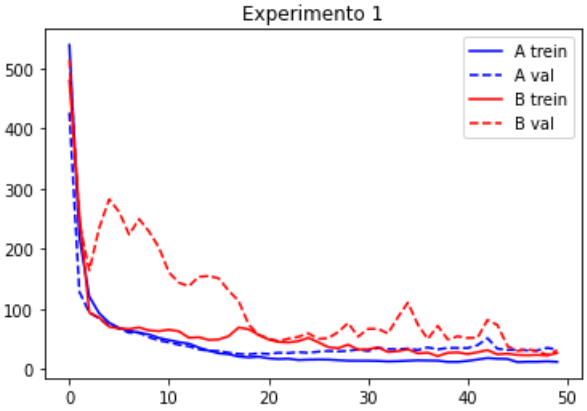
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import boston_housing
from numpy.random import seed
from tensorflow.random import set_seed

(x_train, y_train), (x_test, y_test) = boston_housing.load_data()

#normalizacao max-min 0-1
maxv = x_train.max(axis=0)
minv = x_train.min(axis=0)
x_train = (x_train - minv)/(maxv-minv)

x_test = (x_test - minv)/(maxv-minv)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_
housing.npz
57344/57026 [==============================] - 0s 0us/step
```

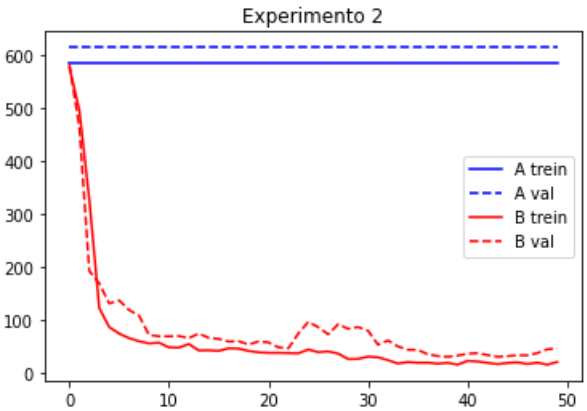
In [2]:
```
def model_A(neurons, layers):
    model1 = keras.Sequential()
    model1.add(keras.layers.Dense(neurons, activation="relu", input_shape=(x_train.shape[
    for l in np.arange(layers-1):
        model1.add(keras.layers.Dense(neurons, activation="relu"))
    model1.add(keras.layers.Dense(1, activation="relu"))
    return model1

def model_B(neurons, layers):
    model2 = keras.Sequential()
    model2.add(keras.layers.Dense(neurons, activation="relu", input_shape=(x_train.shape[

    for l in np.arange((layers//2)-1):
        model2.add(keras.layers.Dense(neurons, activation="relu"))
        model2.add(keras.layers.BatchNormalization())
        model2.add(keras.layers.Dense(neurons, activation="relu"))

    model2.add(keras.layers.Dense(neurons, activation="relu"))
    model2.add(keras.layers.Dense(1, activation="relu"))
    return model2
```

In [3]:
```
epochs=50
batch_size=32

for i in np.arange(1,4):
    print(i, " - modelo A")
    seed(i)
    set_seed(i)
    modelA = model_A(6,12)
    modelA.compile(optimizer=keras.optimizers.Adam(0.01),
                   loss='mse')
    historyA = modelA.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,
                          validation_data=(x_test,y_test), verbose=0)

    print(i, " - modelo B")
    seed(i)
    set_seed(i)
    modelB = model_B(6,12)
    modelB.compile(optimizer=keras.optimizers.Adam(0.01),
                   loss='mse')
    historyB = modelB.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,
                          validation_data=(x_test,y_test), verbose=0)

    plt.plot(historyA.history['loss'], 'b')
    plt.plot(historyA.history['val_loss'], '--b')
    plt.plot(historyB.history['loss'], 'r')
    plt.plot(historyB.history['val_loss'], '--r')
    plt.legend(['A trein','A val', 'B trein','B val'])
    plt.title('Experimento %d' % (i))
    plt.show()
```

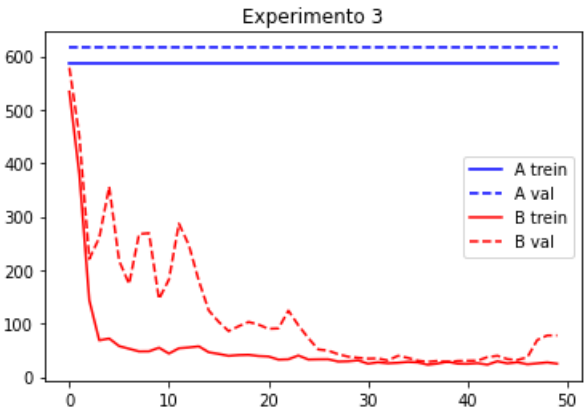
```
1   - modelo A
1   - modelo B
```




```
2   - modelo A
2   - modelo B
```




```
3   - modelo A
3   - modelo B
```



Exercício 7)

Considere a mesma base de dados e condições de treinamento (incluindo a inicialização das sementes). Projete uma nova rede neural com o seguinte formato:

Arquitetura C (incluindo Dropout):

- 1 camada densa com 6 neurônios e ativação `relu`
- 1 camada Batch Normalization
- 10 camadas densas com 6 neurônios e ativação `relu`
- 1 camada Dropout com probabilidade 0.2

- 1 camada densa com 6 neurônios e ativação `relu`
- 1 camada Dropout com probabilidade 0.5
- 1 camada densa com 1 neurônio e ativação `relu`

Trace o gráfico da função de custo computada no treinamento e no teste ao longo das épocas. Podemos dizer sobre o processo de convergência do custo que:

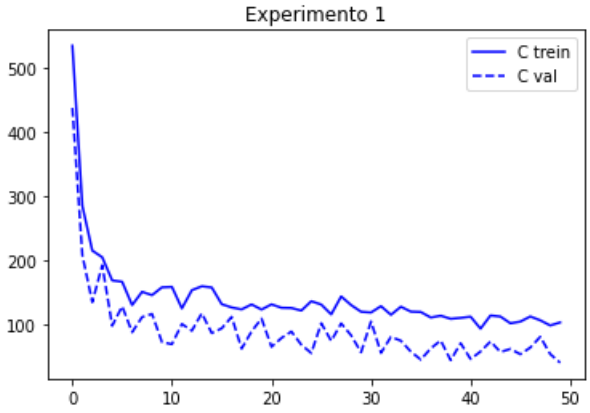
(a) C: não convergiu na maior parte dos experimentos, indicando que Dropout não foi efetivo em auxiliar no treinamento dessa rede profunda
(b) C: apresentou convergência do modelo em todos os casos, com perda na validação similar ou ligeiramente menor do que quando medida no treinamento
(c) C: apresentou convergência do modelo em todos os casos, mas com problemas de generalização, pois a perda na validação aumentou consideravelmente ao longo das épocas, em comparação com a perda medida no treinamento
(d) C: não convergiu em nenhum dos experimentos, indicando que Dropout não foi efetivo em auxiliar no treinamento dessa rede profunda

Justificativa: Ver código abaixo.

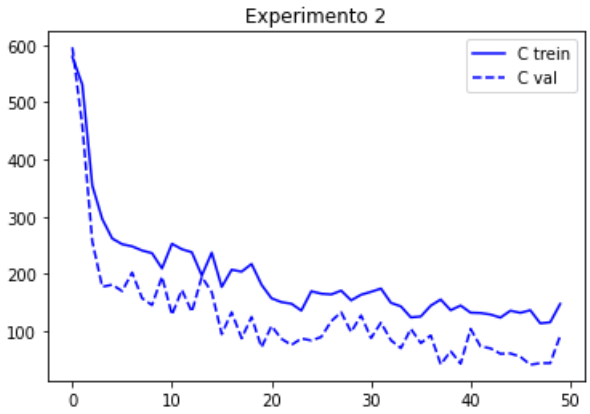
In [4]:
```
def model_C(neurons, layers):
    model3 = keras.Sequential()
    model3.add(keras.layers.Dense(neurons, activation="relu", input_shape=(x_train.shape[
    model3.add(keras.layers.BatchNormalization())
    for l in np.arange(layers-2):
        model3.add(keras.layers.Dense(neurons, activation="relu"))

    model3.add(keras.layers.Dropout(0.2))
    model3.add(keras.layers.Dense(neurons, activation="relu"))
    model3.add(keras.layers.Dropout(0.5))
    model3.add(keras.layers.Dense(1, activation="relu"))
    return model3
```

In [5]:
```
epochs=50
batch_size=32

for i in np.arange(1,4):
    print(i, " - modelo C")
    seed(i)
    set_seed(i)
    modelC = model_C(6,12)
    modelC.compile(optimizer=keras.optimizers.Adam(0.01),
                   loss='mse')
    historyC = modelC.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,
                          validation_data=(x_test,y_test), verbose=0)

    plt.plot(historyC.history['loss'], 'b')
    plt.plot(historyC.history['val_loss'], '--b')
    plt.legend(['C trein','C val'])
    plt.title('Experimento %d' % (i))
    plt.show()

modelC.summary()
```

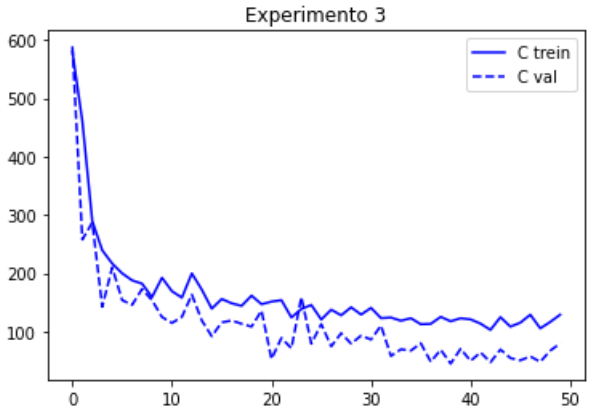
```
1  - modelo C
```




```
2  - modelo C
```




```
3  - modelo C
```




```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_104 (Dense)            (None, 6)                 84
_________________________________________________________________
batch_normalization_17 (Batc (None, 6)                 24
_________________________________________________________________
dense_105 (Dense)            (None, 6)                 42
_________________________________________________________________
dense_106 (Dense)            (None, 6)                 42
_________________________________________________________________
dense_107 (Dense)            (None, 6)                 42
_________________________________________________________________
dense_108 (Dense)            (None, 6)                 42
_________________________________________________________________
dense_109 (Dense)            (None, 6)                 42
_________________________________________________________________
dense_110 (Dense)            (None, 6)                 42
```

dense_111 (Dense)	(None, 6)	42
dense_112 (Dense)	(None, 6)	42
dense_113 (Dense)	(None, 6)	42
dense_114 (Dense)	(None, 6)	42

Exercício 8)

Carregue uma rede existente do pacote Application do Keras, em particular a NASNetMobile (ver: https://keras.io/api/applications/nasnet/#nasnetmobile-function), com pesos aleatórios e pronta para receber imagens da base de dados `cifar10` carregada do TensorFlow Datasets.

Considere o código abaixo que carrega essa base de dados (utilizamos um subconjunto de treinamento e validação), e seu posterior processamento. Redimensionamos as imagens para 128×128 para permitir posterior uso em CNNs pré-treinadas (ver próximo exercício). Note também que é preciso obter codificação one-hot-encoding para as classes pois essa base de dados não é binária. Utilizaremos essa mesma base de dados nos exercícios subsequentes.

Utilize a arquitetura "residual" proposta no notebook 1 da aula, porém altere para que as camadas residuais tenham 32 filtros (ao invés de 64), e a camada convolucional final tenha 256 (ao invés de 512).

Adicione ainda a possibilidade de incluir aumentação de dados como uma camada, logo após a camada de entrada (use um parâmetro augmentation=True/False para ligar e desligar). A aumentação de dados deve conter as seguintes operações (nessa ordem):

- RandomFlip("horizontal")
- RandomContrast(0.3)
- RandomRotation(0.2)

Inicialize as sementes do numpy para 1 e tensorflow para 2, e crie, compile e execute um modelo conforme a função definida, utilizando: Dropout com 0.2, BatchNormalization e Augmentation, bem como: função de custo entropia cruzada categórica, otimizador Adam com taxa de aprendizado 0.001 e 64 exemplos no batch. Compute também a acurácia durante o treinamento.

Qual foi o intervalo de acurácia no treinamento e validação avaliados (use a função `evaluate`) após o treinamento por 5 épocas?

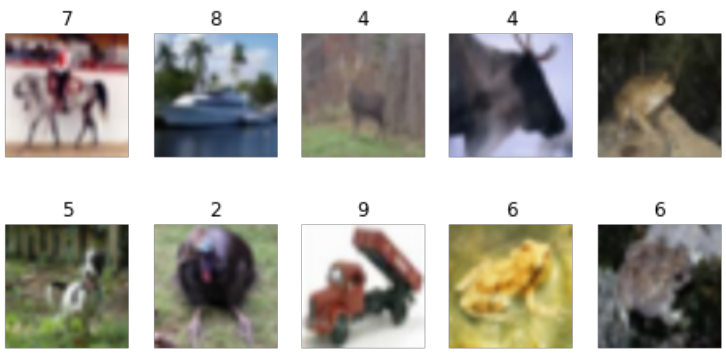
(a) Treinamento=[35,39]; Val=[34,38]
(b) Treinamento=[48,50]; Val=[49,53]
(c) Treinamento=[59,64]; Val=[50,58]
(d) Treinamento=[39,44]; Val=[37,40]

Justificativa: Notar no código abaixo a comparação.

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from numpy.random import seed
from tensorflow.random import set_seed
import tensorflow_datasets as tfds

tfds.disable_progress_bar()

(train_ds, validation_ds), info = tfds.load(
    "cifar10",
    split=["train[0%:30%]", "train[30%:40%]"],
    as_supervised=True,
    with_info=True
)

num_classes = info.features["label"].num_classes
print("Classes: ", num_classes)

def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""
  return tf.cast(image, tf.float32) / 255., label

img_size = (128, 128)
train_ds = train_ds.map(lambda x, y: (tf.image.resize(x, img_size), y))
train_ds = train_ds.map(normalize_img, num_parallel_calls=tf.data.experimental.AUTOTUNE)

validation_ds = validation_ds.map(lambda x, y: (tf.image.resize(x, img_size), y))
validation_ds = validation_ds.map(normalize_img, num_parallel_calls=tf.data.experimental.

input_shape = img_size + (3,)
```

```
Classes:  10
```

In [14]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
for i, (image, label) in enumerate(train_ds.take(10)):
    ax = plt.subplot(2, 5, i + 1)
    plt.imshow(image)
    plt.title(int(label))
    plt.axis("off")
```



In [15]:
```
def label_preprocess(image, label):
    label = tf.one_hot(label, num_classes)
    return image, label

if (num_classes > 2):
    train_ds = train_ds.map(
        label_preprocess, num_parallel_calls=tf.data.experimental.AUTOTUNE
    )
    validation_ds = validation_ds.map(
        label_preprocess, num_parallel_calls=tf.data.experimental.AUTOTUNE
    )
```

In [16]:
```
batch_size = 64
train_ds = train_ds.cache().batch(batch_size).prefetch(buffer_size=10)
validation_ds = validation_ds.cache().batch(batch_size).prefetch(buffer_size=10)
```

In [10]:
```
####################
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomContrast(0.3),
        layers.experimental.preprocessing.RandomRotation(0.2),
    ]
)
```

In [11]:
```
def my_cnn(input_shape, num_classes, dropout_rate=0.0, batch_norm=False, augmentation=Fa

    inputs = keras.Input(shape=input_shape)

    if (augmentation):
        x = data_augmentation(inputs)
        x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    else:
        x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)

    if (batch_norm): x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)

    # guarda ativacao para somar ao fim do bloco residual
    ativacao_residual = x
    for n_filtros in [32,32]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(n_filtros, 3, padding="same")(x)
        if (batch_norm): x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(n_filtros, 3, padding="same")(x)
        if (batch_norm): x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # residual
        residual = layers.Conv2D(n_filtros, 1, strides=2, padding="same")(
            ativacao_residual
        )
        x = layers.add([x, residual])  # adiciona resisual
        ativacao_residual = x  # armazena saida do bloco

    x = layers.SeparableConv2D(256, 3, padding="same")(x)
    if (batch_norm): x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        neuronios = 1
    else:
        activation = "softmax"
        neuronios = num_classes

    x = layers.Dropout(dropout_rate)(x)
    outputs = layers.Dense(neuronios, activation=activation)(x)

    return keras.Model(inputs, outputs)
```

In [17]:
```
seed(1)
set_seed(2)
epochs = 5
model_aug = my_cnn(input_shape, num_classes, dropout_rate=0.2, batch_norm=True, augmentat
model_aug.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

hist_aug = model_aug.fit(train_ds, batch_size=batch_size,
                   epochs=epochs, validation_data=validation_ds, verbose=1)
```

```
Epoch 1/5
      2/Unknown - 0s 87ms/step - loss: 2.5324 - accuracy: 0.0547WARNING:tensorflow:Callba
cks method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0366s v
s `on_train_batch_end` time: 0.0688s). Check your callbacks.
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0366s vs `on_train_batch_end` time: 0.0688s). Check your callbacks.
235/235 [==============================] - 25s 106ms/step - loss: 1.9316 - accuracy: 0.28
42 - val_loss: 2.9122 - val_accuracy: 0.0978
Epoch 2/5
235/235 [==============================] - 21s 90ms/step - loss: 1.7800 - accuracy: 0.348
8 - val_loss: 2.6413 - val_accuracy: 0.1874
Epoch 3/5
235/235 [==============================] - 21s 89ms/step - loss: 1.7078 - accuracy: 0.377
3 - val_loss: 1.7431 - val_accuracy: 0.3618
Epoch 4/5
235/235 [==============================] - 21s 90ms/step - loss: 1.6579 - accuracy: 0.396
5 - val_loss: 1.6495 - val_accuracy: 0.3988
Epoch 5/5
235/235 [==============================] - 21s 90ms/step - loss: 1.6082 - accuracy: 0.421
5 - val_loss: 1.7832 - val_accuracy: 0.3642
```

In [24]:
```
scoreTre = model_aug.evaluate(train_ds, verbose=0)
scoreVal = model_aug.evaluate(validation_ds, verbose=0)

print("Pesos aleatórios + data augmentation (Acurácia):")
print("Treinamento: %.4f" % (scoreTre[1]))
print("Validação  : %.4f" % (scoreVal[1]))
```

```
Pesos aleatórios + data augmentation (Acurácia):
Treinamento: 0.3730
Validação  : 0.3642
```

Exercício 9)

Utilize a mesma base de dados, conforme carregada e pré-processada no exercício anterior.

Carregue como modelo base a CNN MobileNetV2 (ver https://keras.io/api/applications/) e seus pesos pré-treinados na ImageNet, sendo preparada para receber como entrada imagens no formato da Cifar10.

Monte um modelo que possua logo após a entrada, a MobileNetV2 (sem data augmentation), seguida de GlobalAveragePooling2D, Dropout de 0.2 e uma camada densa com ativação Softmax. Torne o modelo base (MobileNetV2) não treinável, treinando apenas a camada Softmax.

Inicialize as sementes do numpy para 1 e tensorflow para 2, compile e treine o modelo.

Qual foi o intervalo de acurácia no treinamento e validação após o treinamento durante 6 épocas?

(a) Treinamento=[76,82]; Val=[74,78]
(b) Treinamento=[42,48]; Val=[33,39]
(c) Treinamento=[70,78]; Val=[65,70]
(d) Treinamento=[59,64]; Val=[32,38]

Justificativa: Notar no código abaixo a comparação.

In [26]:
```
epochs = 5

base_model = keras.applications.MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=input_shape,
)
base_model.trainable = False

# Nosso modelo
inputs = keras.Input(shape=input_shape)
x = base_model(inputs, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)
outputs = keras.layers.Dense(num_classes, "softmax")(x) # essa será a única camada treina
model = keras.Model(inputs, outputs)
model.summary()

seed(1)
set_seed(2)
model.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

histMob = model.fit(train_ds, batch_size=batch_size,
                    epochs=epochs, validation_data=validation_ds, verbose=1)
```

```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
mobilenetv2_1.00_128 (Functi (None, 4, 4, 1280)        2257984
_________________________________________________________________
global_average_pooling2d_4 ( (None, 1280)              0
_________________________________________________________________
dropout_10 (Dropout)         (None, 1280)              0
_________________________________________________________________
dense_121 (Dense)            (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
_________________________________________________________________
Epoch 1/5
235/235 [==============================] - 8s 34ms/step - loss: 1.2094 - accuracy: 0.5842
- val_loss: 0.8259 - val_accuracy: 0.7114
Epoch 2/5
235/235 [==============================] - 7s 31ms/step - loss: 0.8099 - accuracy: 0.7178
- val_loss: 0.7393 - val_accuracy: 0.7412
Epoch 3/5
235/235 [==============================] - 7s 30ms/step - loss: 0.7339 - accuracy: 0.7409
- val_loss: 0.7013 - val_accuracy: 0.7546
Epoch 4/5
235/235 [==============================] - 7s 30ms/step - loss: 0.6973 - accuracy: 0.7576
- val_loss: 0.6915 - val_accuracy: 0.7532
Epoch 5/5
235/235 [==============================] - 7s 30ms/step - loss: 0.6622 - accuracy: 0.7701
- val_loss: 0.6775 - val_accuracy: 0.7574
```

In [27]:
```
scoreTre = model.evaluate(train_ds, verbose=0)
scoreVal = model.evaluate(validation_ds, verbose=0)
print("Pesos pré-treinados + treinamento classificador (Acurácia):")
print("Treinamento: %.4f" % (scoreTre[1]))
print("Validação  : %.4f" % (scoreVal[1]))
```

```
Pesos pré-treinados + treinamento classificador (Acurácia):
Treinamento: 0.8067
Validação  : 0.7574
```

Exercício 10)

Similar ao exercício anterior, monte um modelo que possua logo após a entrada, a MobileNetV2 (sem data augmentation), seguida de GlobalAveragePooling2D, Dropout de 0.2 e uma camada densa com ativação Softmax. Torne o modelo base (MobileNetV2) treinável. Observação: recarregue os pesos, não use os pesos utilizados a partir do exercício anterior.

Qual foi o intervalo de acurácia no treinamento e validação após o treinamento durante 6 épocas?

(a) Treinamento=[59,65]; Val=[70,76]
(b) Treinamento=[75,84]; Val=[75,78]
(c) Treinamento=[88,90]; Val=[32,38]
(d) Treinamento=[92,97]; Val=[78,87]

Justificativa: Notar no código abaixo a comparação.

In [28]:
```
epochs = 5
base_model2 = keras.applications.MobileNetV2(
    weights="imagenet",
    include_top=False,
    input_shape=input_shape,
)

base_model2.trainable = True

# Nosso modelo
inputs = keras.Input(shape=input_shape)
x = base_model2(inputs, training=True)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)
outputs = keras.layers.Dense(num_classes, "softmax")(x)
model2 = keras.Model(inputs, outputs)
model2.summary()

seed(1)
set_seed(2)
model2.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

histMob2 = model2.fit(train_ds, batch_size=batch_size,
                    epochs=epochs, validation_data=validation_ds, verbose=1)
```

```
Model: "functional_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
mobilenetv2_1.00_128 (Functi (None, 4, 4, 1280)        2257984
_________________________________________________________________
global_average_pooling2d_5 ( (None, 1280)              0
_________________________________________________________________
dropout_11 (Dropout)         (None, 1280)              0
_________________________________________________________________
dense_122 (Dense)            (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 2,236,682
Non-trainable params: 34,112
_________________________________________________________________
Epoch 1/5
      2/Unknown - 0s 89ms/step - loss: 2.4647 - accuracy: 0.2031WARNING:tensorflow:Callba
cks method `on_train_batch_end` is slow compared to the batch time (batch time: 0.0319s v
s `on_train_batch_end` time: 0.0706s). Check your callbacks.
```

```
WARNING:tensorflow:Callbacks method `on_train_batch_end` is slow compared to the batch ti
me (batch time: 0.0319s vs `on_train_batch_end` time: 0.0706s). Check your callbacks.
235/235 [==============================] - 26s 110ms/step - loss: 0.6692 - accuracy: 0.78
75 - val_loss: 0.5129 - val_accuracy: 0.8314
Epoch 2/5
235/235 [==============================] - 25s 107ms/step - loss: 0.3349 - accuracy: 0.88
76 - val_loss: 0.5024 - val_accuracy: 0.8424
Epoch 3/5
235/235 [==============================] - 25s 108ms/step - loss: 0.2520 - accuracy: 0.91
54 - val_loss: 0.5535 - val_accuracy: 0.8444
Epoch 4/5
235/235 [==============================] - 25s 108ms/step - loss: 0.2036 - accuracy: 0.93
17 - val_loss: 0.5192 - val_accuracy: 0.8456
Epoch 5/5
235/235 [==============================] - 25s 107ms/step - loss: 0.1665 - accuracy: 0.94
21 - val_loss: 0.6161 - val_accuracy: 0.8516
```

In [29]:

```
scoreTre = model2.evaluate(train_ds, verbose=0)
scoreVal = model2.evaluate(validation_ds, verbose=0)
print("Pesos pré-treinados + fine-tuning (Acurácia):")
print("Treinamento: %.4f" % (scoreTre[1]))
print("Validação  : %.4f" % (scoreVal[1]))
```

```
Pesos pré-treinados + fine-tuning (Acurácia):
Treinamento: 0.9455
Validação  : 0.8516
```