Trabalho #1 - Biblioteca de dados, RNAs pré-treinadas e PipeLine de dados

Nesse trabalho você vai criar e treinar uma RNA, usando como base uma rede pré-treinada, para resolver um problema de classificação multiclasse.

Para fazer isso você vai criar um pipeline de dados para treinamento, usar os vetores de características da RNA EfficientNet e um conjunto de dados de imagens de satélite para treinar uma nova RNA para classificar tipos de áreas a partir de imagens de satélites.

Esse trabalho pede ser realizado por equipes de até 3 alunos.

Coloque o seu nome aqui:

Nome: Gabriel Silvestre Mancini

Nome:

Nome:

1. Importar bibliotecas

Primeiramente devemos instalar o tf_keras, que é uma versão compatível do Keras com o TensorFlow 2.17 para permitir usar alguns modelos do TensorFlow Hub.

Execute as células abaixo para importar as bilbiotecas necessárias.

```
!pip install tf_keras
```

```
Requirement already satisfied: tf_keras in /usr/local/lib/python3.10/dist-packages (2
Requirement already satisfied: tensorflow<2.18,>=2.17 in /usr/local/lib/python3.10/di
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/
```

```
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/d
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/li
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages
```

```
import tensorflow as tf
import tf_keras as keras
print("Using TensorFlow Version:", tf.__version__)
print("Using Keras Version:", keras.__version__)

import tensorflow_datasets as tfds
import tensorflow_hub as hub

import matplotlib.pyplot as plt
import numpy as np
```

```
Using TensorFlow Version: 2.17.1
Using Keras Version: 2.17.0
```

2. Carregar dados do TensorFlow Data Service (TFDS)

O conjunto de dados EuroSAT é baseado nas imagens do satélite Sentinel-2 e consiste de 27.000 imagens coloridas de dimensão 64x64x3 com 10 classes.

Dois conjuntos de dados são disponibilizados: (1) eurosat/rgb que contém imagens no formato RGB, que é o que usaremos; e (2) eurosat/all com imagens de 13 canais diferentes.

Exercício #1: Carregar dados

A primeira etapa é carregar as imagens. Esse conjunto de dados está disponível no TFDS com o nome `eurosat/rgb`. Os dados estão em um único conjunto de nome "train" e, portanto precisam ser divididos em pelo menos dois conjuntos: dados de treinamento e de validação.

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://www.tensorflow.org/datasets.

Ao carregar os dados, use o argumento `split` com porcentagens para separar os dados em dois conjuntos: dados de treinamento (80% dos dados) e de validação (20% dos dados). Para obter maiores detalhes de como usar o método `tfds.load` pode ser obtido em https://www.tensorflow.org/datasets/splits.

```
# Carrega dados do TFDS
# Inclua seu código aqui (2 linhas)
train_data, info = tfds.load('eurosat/rgb', shuffle_files=True, as_supervised=True, with_
val_data = tfds.load('eurosat/rgb', as_supervised=True, split="train[80%:]")
```

Execute a célula abaixo para visualizar as informações sobre esse o conjunto de dados `eurosat/rgb`.

```
info
```

```
tfds.core.DatasetInfo(
    name='eurosat',
    full_name='eurosat/rgb/2.0.0',
    description="""
    EuroSAT dataset is based on Sentinel-2 satellite images covering 13 spectral
    bands and consisting of 10 classes with 27000 labeled and
    geo-referenced samples.

    Two datasets are offered:
    - rgb: Contains only the optical R, G, B frequency bands encoded as JPEG image.
    - all: Contains all 13 bands in the original value range (float32).

    URL: https://github.com/phelber/eurosat
    """,
    config_description="""
    Sentinel-2 RGB channels
    """,
    homepage='https://github.com/phelber/eurosat',
    data_dir='/root/tensorflow_datasets/eurosat/rgb/2.0.0',
    file_format=tfrecord,
    download_size=89.91 MiB,
    dataset_size=89.50 MiB,
    features=FeaturesDict({
        'filename': Text(shape=(), dtype=string),
        'image': Image(shape=(64, 64, 3), dtype=uint8),
        'label': ClassLabel(shape=(), dtype=int64, num_classes=10),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'train': <SplitInfo num_examples=27000, num_shards=1>,
    },
    citation="""@misc{helber2017eurosat,
        title={EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and
Land Cover Classification},
        author={Patrick Helber and Benjamin Bischke and Andreas Dengel and Damian
Borth},
        year={2017},
        eprint={1709.00029},
```

```
            archivePrefix={arXiv},
            primaryClass={cs.CV}
        }""",
    )
```

Verifique o número de exemplos de treinamento e de validação executando a célula abaixo.

```
print('Número exemplos de treinamento =', len(list(train_data)))
print('Número exemplos de validação =', len(list(val_data)))
```

```
Número exemplos de treinamento = 21600
Número exemplos de validação = 5400
```

Saída esperada:

```
Número exemplos de treinamento = 21600
Número exemplos de treinamento = 5400
```

Execute a célula abaixo para definir a lista com os nomes das classes existentes no conjunto de dados.

```
labels_list = ['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway', 'Industrial',
               'Pasture', 'PermanentCrop', 'Residential', 'River', 'SeaLake']
print(labels_list)
```

```
['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway', 'Industrial', 'Pasture',
```

Exercíco #2: Visualização dos dados

Na célula abaixo escreva um código para obter 5 exemplos dos dados de treinamento e visualizá-los juntamento com os nomes (`labels_list`) e números das classes.

Para fazer isso você vai precisar de um laço `for`, dos comandos `print`, `plt.imshow()`, `plt.show()`, e do método `take()`.

```
# Itera no conjunto de dados pegando exemplos
# Inclua seu código aqui (~5 linhas)
# Cria o plot para 9 imagens
plt.figure(figsize=(10, 10))

for i, (image, label) in enumerate(train_data.take(5)):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image)
    plt.title(labels_list[label.numpy()])
    plt.axis("off")
```

```
plt.show()
```

Exemplo de saída esperada (sem as figuras):

```
Classe: Forest - 1
Classe: Residential - 7
Classe: River - 8
```

3. Processamento dos dados

Após carregar os dados, você tem que processá-los para poderem ser usados pela RNA.

O módulo de vetores de características da EfficientNet permite que as imagens tenham em princípio qualquer dimensão, assim, não é necessário redimensionar as imagens. Porém, os valores dos pixels devem estar entre 0 e 1, conforme mencionado nas instruções de uso desse módulo, que podem ser vistas em https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1.

Exercício #3: Normalização das imagens

Na célula abaixo escreva um código para normalizar um lote de imagens e depois crie os lotes de dados de treinamento e de validação.

Conforme vimos na aula, ao importar os dados do TF Data Services, os dados são armazenados em objetos e para podermos usar esses dados de forma eficiente temos que usar os métodos fornecidos para esse tipo de objeto.

Para normalizar e redimensionar as imagens crie uma função de nome `format_image` e a utilize chamando o método `map()`.

A dimensão das imagens deve ser 64x64 pixels.

Para normalizar os dados você vai precisar primeiramente transformá-los em `float32`, para isso use a função `tf.cast()`. As instruções de uso dessa função podem ser vistas em https://www.tensorflow.org/api_docs/python/tf/cast.

Crie um dataset eficiente, para isso utilize processamento em paralelo quando executar a função de pré-processamento dos dados. Use tamném os métodos `cache` e `prefetch`.

```
# Definição da dimensão das imagens para processamento e do tamanho dos lotes de dados

# Use otimização do pipeline
AUTOTUNE = tf.data.AUTOTUNE

# Função usada para redimensionar e normalizar as imagens
# Inclua seu código aqui (~3 linhas)
def format_image(image, label):
    image = tf.image.resize(image, [64, 64])
    image = tf.cast(image, tf.float32) / 255.0
    return image, label

# Cria lotes de dados usando o método map() para chamar a função format_image()
# Inclua seu código aqui (~2 linhas)
# Cria lotes de dados e aplica cache e prefetch
train_batches = train_data.map(format_image, num_parallel_calls=AUTOTUNE).cache().batch(3
val_batches = val_data.map(format_image, num_parallel_calls=AUTOTUNE).cache().batch(32).p

train_batches
```

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 64, 64, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int64, name=None))>
```

Saída espeada:

```
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 64, 64, 3), dtype=tf.float32, name=None),
```

Execute a célula abixo para verificar se o seu dataset está correto.

```
for img, label in train_batches.take(1):
    print('Dimensão de um lote de imagens:', img.shape)
    print('Dimensão de um lote de saídas desejadas:', label.shape)
```

```
Dimensão de um lote de imagens: (32, 64, 64, 3)
Dimensão de um lote de saídas desejadas: (32,)
```

Saída esperada:

```
Dimensão de um lote de imagens: (32, 64, 64, 3)
Dimensão de um lote de saídas desejadas: (32,)
```

4. Criação da RNA

Nesse trabalho você vai criar e treinar uma RNA para identificar tipos de áreas a partir de imagens do satélite Sentinel-2.

Para isso você vai criar uma RNA usando como base o módulo de vetor de características da rede EfficientNet, que foi treinada com as imagens da ImageNet.

As redes EfficientNets são utilizadas para classificar imagens e apresentam um desempenho similar a outras redes mais conhecidas, porém, possui um número muito menor de parâmetros e é muito mais rápida. O trabalho que originou essa RNA é Mingxing Tan and Quoc V. Le: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, ICML 2019.

Exercício #4: Carregar vetor de características do TF-Hub

Na célula abaixo crie um código que importa o módulo de vetores de características da EfficientNet e o coloca no objeto `feature_extractor`. Essa rede estão no TF-Hub e as informações de como usá-la podem ser obtidas no link https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1.

Pra fazer isso você vai precisar definir a dimensão do tensor de entrada da rede usando o argumento `input_shape` e a função `hub.KerasLayer()`.

```
# Dimensão das imagens para argumento input_shape
# Inclua seu código aqui (~1 linha)
input_shape = (64, 64, 3)

# Carrega vetores de características com a URL do módulo
# Inclua seu código aqui (~2 linhas)
feature_extractor = hub.KerasLayer(
```

```
    "https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1",
    input_shape=input_shape,
)

feature_extractor
```

```
<tensorflow_hub.keras_layer.KerasLayer at 0x7bc0ec11e770>
```

Saída espeada:

```
<tensorflow_hub.keras_layer.KerasLayer at 0x7ff58df7ae48>
```

Execute a célula abaixo para verificar os seus resultados.

```
for img, label in train_batches.take(2):
    feat = feature_extractor(img)
    print(feat.shape)
```

```
(32, 1280)
(32, 1280)
```

Saída esperada:

```
(32, 1280)
(32, 1280)
```

Exercício #5: Criação da RNA como o Keras

Na célula abaixo crie um código que incorpora o `feature_extractor`, criado no exercício #4, em uma rede sequencial do Keras para realizar a tarefa de classificação multiclasse com 10 classes.

Após criar a RNA utilize o método `summary()` para apresentá-la.

```
# Número de classes da RNA
# Inclua seu código aqui (~1 linha)
NUM_CLASSES = 10

# Cria modelo seguencial do Keras para problema de classificação com 10 classes
# Inclua seu código aqui (~3 linhas)
model = keras.models.Sequential([
       keras.layers.InputLayer(input_shape=input_shape),
       feature_extractor,
       keras.layers.Dense(NUM_CLASSES, activation='softmax')
])
```

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',  # Perda para classificação multiclasse
    metrics=['accuracy']  # Métrica de acurácia
)

# Apresenta configuração da RNA
# Inclua seu código aqui (~1 linha)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              4049564

 dense (Dense)               (None, 10)                12810

=================================================================
Total params: 4062374 (15.50 MB)
Trainable params: 12810 (50.04 KB)
Non-trainable params: 4049564 (15.45 MB)
_________________________________________________________________
```

Saída esperada:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              4049564
_________________________________________________________________
dense (Dense)                (None, 10)                12810
=================================================================
Total params: 4,062,374
Trainable params: 12,810
Non-trainable params: 4,049,564
_________________________________________________________________
```

5. Compilação e treinamento da RNA

Como visto em aula, o treinamento da RNA deve ser realizado de forma que somente os parâmetros da camada densa, adicionada ao extrator de características, sejam alterados durante o treinamento. Isso é necessário para não destrui a parte da RNA que corresponde à EfficientNet, que já foi previamente treinada com um conjunto de centenas de milhares de imagens. Assim, você tem que "congelar" os parâmetros do extrator e características.

Exercício #6: Compilação da RNA

Na célula abaixo crie um código que congela os parâmetros do `feature_extractor` e compila a RNA usando os seguintes parâmetros:

- Método de otimização: Adam;
- Fução de custo: `sparse_categorical_crossentropy`;
- Métrica: `accuracy`.

```
# Congela parâmetros da MobiliNet
# Inclua seu código aqui (~1 linha)
feature_extractor.trainable = False

# Define método de otimização
# Inclua seu código aqui (~1 linha)
otimizador = keras.optimizers.Adam()

# Compila RNA
# Inclua seu código aqui (~1 comando)
model.compile(
    optimizer=otimizador,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Exercício #7: Treinamento da RNA

O treinamento da RNA deve ser realizado com o método `fit` e os dados de treinamento e validação são fornecidos por meio dos objetos `train_batches` e `val_batches`.

Na célula abaixo crie um código que realiza o treinameto da RNA usando 30 épocas de treinamento.

```
# Define número de épocas de treinamento
# Inclua seu código aqui (~1 linha)
epocas = 30

# Realiza o treinamento usando os dados de treinamento e validação
# Inclua seu código aqui (~1 comando)
history = model.fit(train_batches, validation_data=val_batches, epochs=epocas)
```

```
Epoch 2/30
675/675 [==============================] - 9s 14ms/step - loss: 0.2656 - accuracy:
Epoch 3/30
675/675 [==============================] - 8s 12ms/step - loss: 0.2235 - accuracy:
Epoch 4/30
```

```
675/675 [==============================] - 9s 14ms/step - loss: 0.1813 - accuracy:
Epoch 6/30
675/675 [==============================] - 8s 12ms/step - loss: 0.1678 - accuracy:
Epoch 7/30
675/675 [==============================] - 9s 13ms/step - loss: 0.1568 - accuracy:
Epoch 8/30
675/675 [==============================] - 9s 13ms/step - loss: 0.1476 - accuracy:
Epoch 9/30
675/675 [==============================] - 8s 13ms/step - loss: 0.1397 - accuracy:
Epoch 10/30
675/675 [==============================] - 9s 13ms/step - loss: 0.1328 - accuracy:
Epoch 11/30
675/675 [==============================] - 9s 14ms/step - loss: 0.1267 - accuracy:
Epoch 12/30
675/675 [==============================] - 9s 14ms/step - loss: 0.1212 - accuracy:
Epoch 13/30
675/675 [==============================] - 8s 12ms/step - loss: 0.1162 - accuracy:
Epoch 14/30
675/675 [==============================] - 9s 14ms/step - loss: 0.1117 - accuracy:
Epoch 15/30
675/675 [==============================] - 9s 14ms/step - loss: 0.1076 - accuracy:
Epoch 16/30
675/675 [==============================] - 8s 12ms/step - loss: 0.1038 - accuracy:
Epoch 17/30
675/675 [==============================] - 9s 14ms/step - loss: 0.1003 - accuracy:
Epoch 18/30
675/675 [==============================] - 9s 14ms/step - loss: 0.0970 - accuracy:
Epoch 19/30
675/675 [==============================] - 9s 14ms/step - loss: 0.0939 - accuracy:
Epoch 20/30
675/675 [==============================] - 8s 12ms/step - loss: 0.0910 - accuracy:
Epoch 21/30
675/675 [==============================] - 9s 14ms/step - loss: 0.0883 - accuracy:
Epoch 22/30
675/675 [==============================] - 10s 14ms/step - loss: 0.0858 - accuracy:
Epoch 23/30
675/675 [==============================] - 8s 12ms/step - loss: 0.0833 - accuracy:
Epoch 24/30
675/675 [==============================] - 9s 13ms/step - loss: 0.0810 - accuracy:
Epoch 25/30
675/675 [==============================] - 10s 14ms/step - loss: 0.0789 - accuracy:
Epoch 26/30
675/675 [==============================] - 10s 15ms/step - loss: 0.0768 - accuracy:
Epoch 27/30
675/675 [==============================] - 8s 12ms/step - loss: 0.0748 - accuracy:
Epoch 28/30
675/675 [==============================] - 9s 14ms/step - loss: 0.0729 - accuracy:
Epoch 29/30
675/675 [==============================] - 9s 14ms/step - loss: 0.0711 - accuracy:
Epoch 30/30
```

Saída esperada:

```
Epoch 1/10
675/675 [==============================] - 13s 19ms/step - loss: 0.5030 - accuracy: 0.8542 - val_l
.
.
```

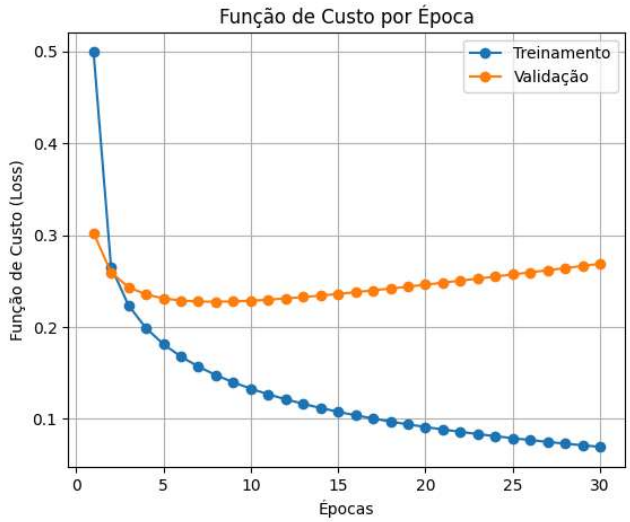
```
.
Epoch 30/30
675/675 [==============================] - 8s 11ms/step - loss: 0.0695 - accuracy: 0.9830 - val_lc
```

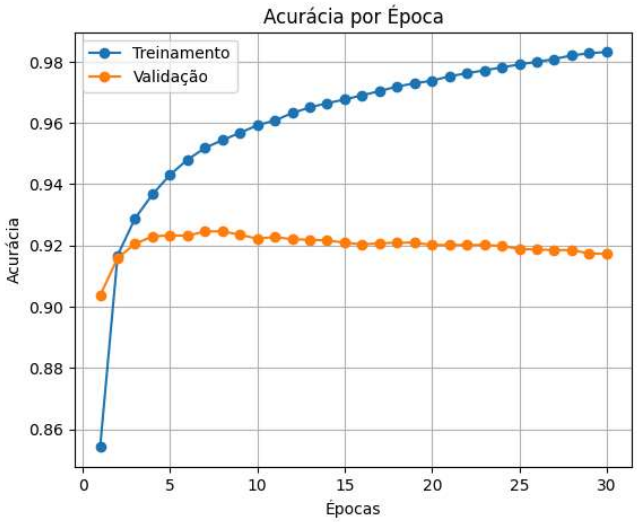
Exercício #8: Resultados do treinamento

Na célula abaixo crie um código que apresenta os resultados do treinamento em função das épocas. Você deve fazer dois gráficos:

1. Valores da função de custo para os dados de treinamento e de validação;
2. Valores da métrica para os dados de treinamento e de validação.

```
# Definir vetores com valores da função de custo e da métrica para os dados de treinament
# Inclua seu código aqui (~5 linhas)
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Criar vetor de épocas
# Inclua seu código aqui (~1 linha)
epochs = range(1, len(train_loss) + 1)

# Fazer o gráfico dos valores da função de custo
# Inclua seu código aqui (~6 linhas)
# Gráfico da função de custo
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)  # Subplot 1: Função de custo
plt.plot(epochs, train_loss, label='Treinamento', marker='o')
plt.plot(epochs, val_loss, label='Validação', marker='o')
plt.title('Função de Custo por Época')
plt.xlabel('Épocas')
plt.ylabel('Função de Custo (Loss)')
plt.legend()
plt.grid(True)


# Fazer o gráfico dos valores da métrica
# Inclua seu código aqui (~6 linhas)
plt.subplot(1, 2, 2)  # Subplot 2: Métrica de acurácia
plt.plot(epochs, train_accuracy, label='Treinamento', marker='o')
plt.plot(epochs, val_accuracy, label='Validação', marker='o')
plt.title('Acurácia por Época')
plt.xlabel('Épocas')
plt.ylabel('Acurácia')
plt.legend()
plt.grid(True)
```





6.4 Teste da RNA

Após o treinamento é necessário verificar o desempenho da RNA. paar isso vamos calcular os valores da função de custo e da métrica para as imagens do conjunto de validação e depois vamos usar o método `predict` para prever as classes de algumas imagens.

O código da célula abaixo calcula o resultado da função de custo e da exatidão para os exemplos validação usando o método `evaluate`.

Exercício #9: Avaliação do desempenho da RNA

Na célula abaixo determine o desemepnho da RNA usando o método `evaluate` para calcular o valor da função d ecusto e da métrica para os dados de validação.

```
# Avalia desempenho da RNA para os dados de validação
# Inclua seu código aqui (~1 linha)
eval_results = model.evaluate(val_batches, verbose=0)

# Apresenta os resultados
# Inclua seu código aqui (~2 linhas)
for metric, value in zip(model.metrics_names, eval_results):
    print(metric + ': {:.4}'.format(value))
```

```
loss: 0.2688
accuracy: 0.9172
```

Saída esperada:

```
loss: 0.2684
accuracy: 0.9181
```

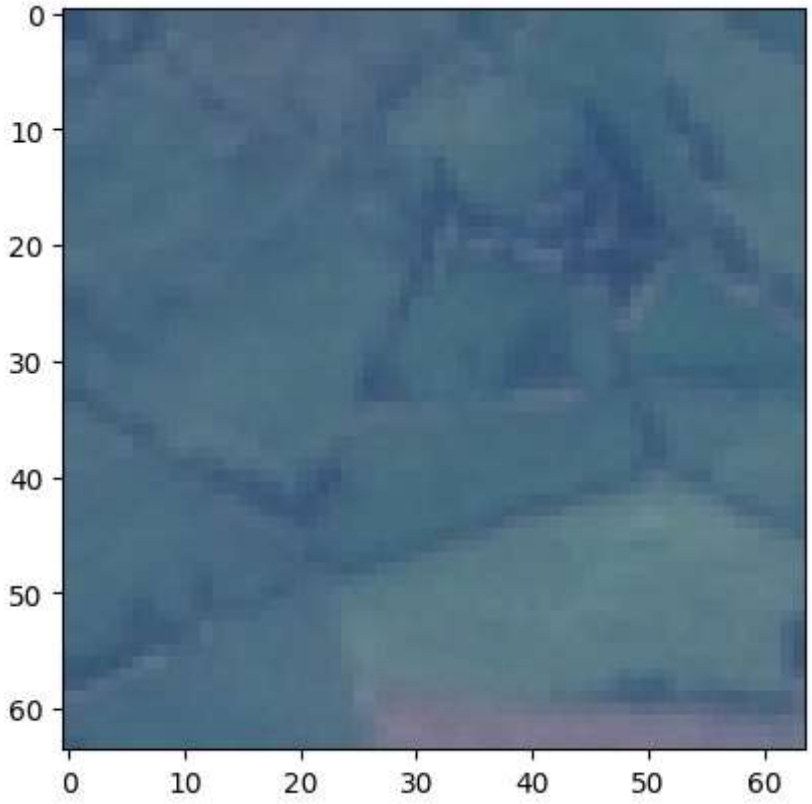
Exercício #10: Teste de classificação de imagens

Para poder fornecer as imagens para a RNA usando o método `predict` você precisa extrai-las do objeto `val_data` e processá-las com a função `format_image`, que por sua vez é chamada pelo método `map()`. Além disso, você tem que incluir o eixo dos exemplos na imagem de acordo com o esperado por uma RNA do Keras.

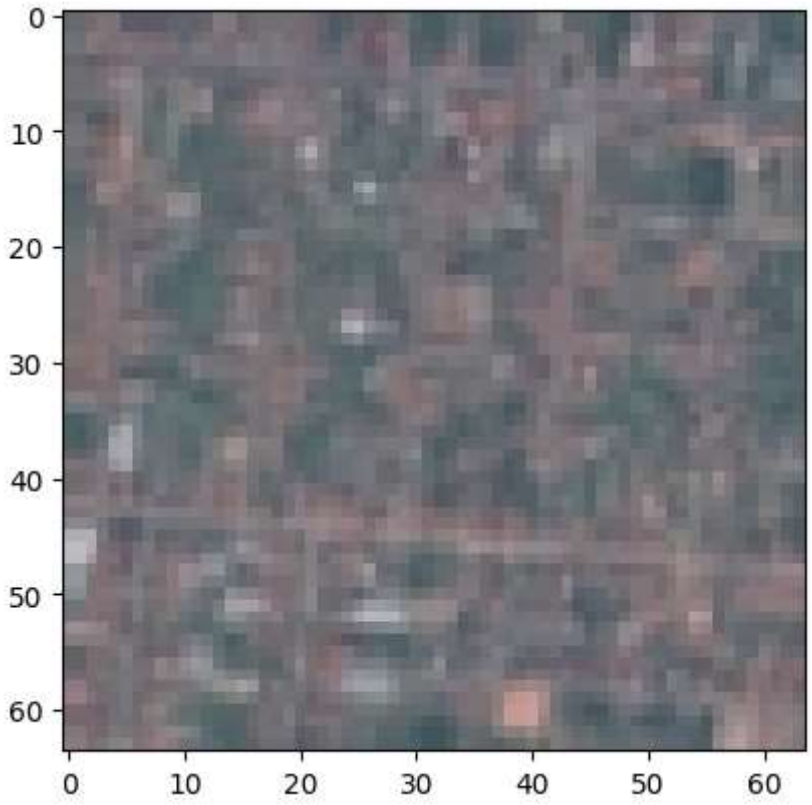
Na célula baixo crie um código que calcula as classes previstas para os 5 primeiros exemplos do conjunto de validação usando o método `predict` e apresenta os resultados junto com as imagens e as classes previstas e reais.

```
# Itera no objeto val_data para pegar 5 imagens e aplica função format_image
for data in val_data.map(format_image).take(5):

    # Extrai imagem e classe prevista
    # Inclua seu código aqui (~1 linha)
    image, label = data

    # Adiciona eixo dos exemplos
    # Inclua seu código aqui (~1 linha)
    image = np.expand_dims(image, axis=0)

    # Calcula probabilidades previstas pela RNA
    # Inclua seu código aqui (~1 linha)
    yprev = model.predict(image)

    # Determina classe prevista
    # Inclua seu código aqui (~1 linha)
    label_prev = np.argmax(yprev)

    # Apresenta resultados das classes e mostra imagem
    # Inclua seu código aqui (~4 linhas)
    print('Classe prevista =', labels_list[label_prev], ', Classe real =', labels_list[la
    plt.imshow(image[0])
    plt.show()
```

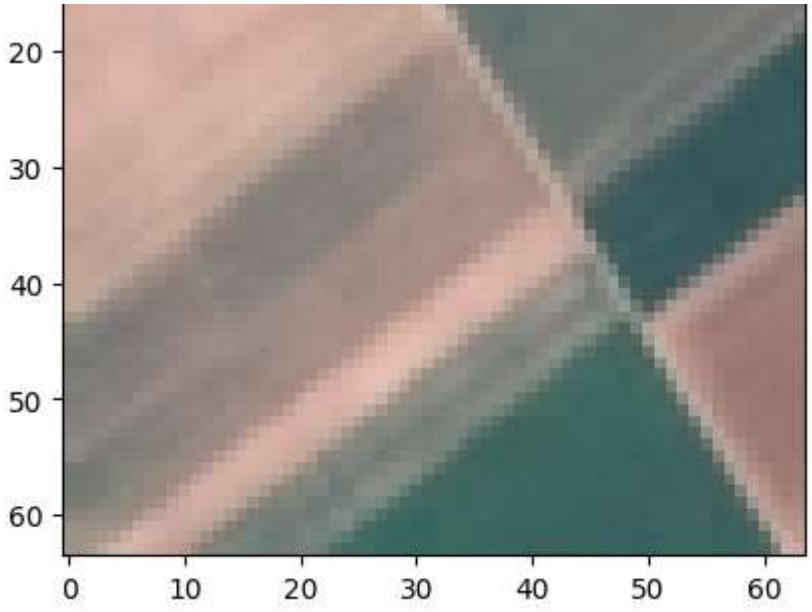
```
1/1 [==============================] - 3s 3s/step
Classe prevista = Pasture , Classe real = Pasture
```



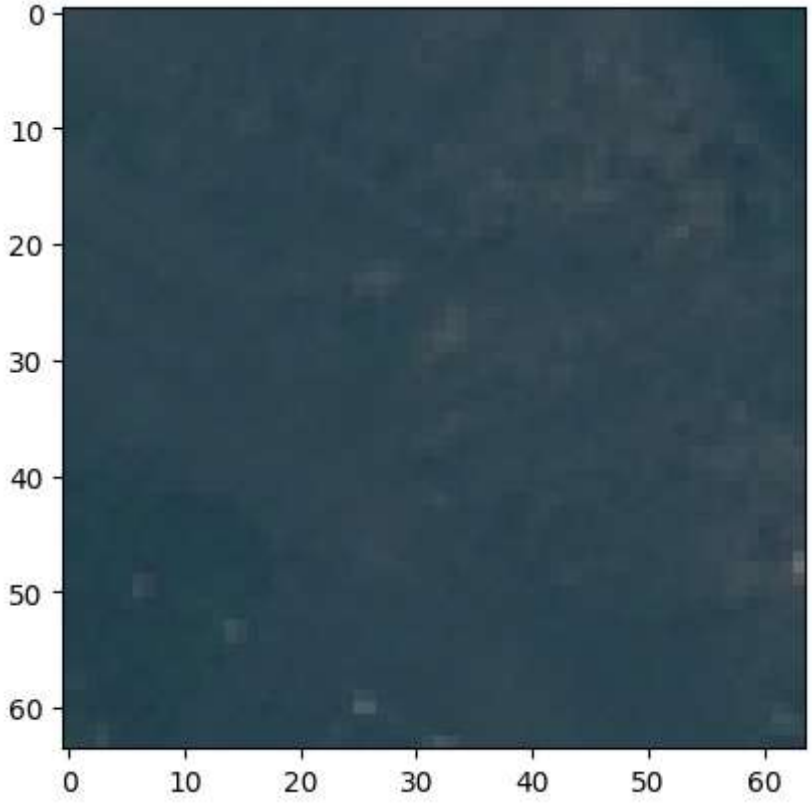

```
1/1 [==============================] - 0s 48ms/step
Classe prevista = Residential , Classe real = Residential
```




```
1/1 [==============================] - 0s 56ms/step
Classe prevista = AnnualCrop , Classe real = AnnualCrop
```




```
1/1 [==============================] - 0s 48ms/step
Classe prevista = Forest , Classe real = Forest
```




```
1/1 [==============================] - 0s 38ms/step
Classe prevista = AnnualCrop , Classe real = AnnualCrop
```