T1_Datasets_Modelos

March 4, 2021

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

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 usará 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.

Nome: Bruno Rodrigues Silva

1 Importar bibliotecas

Execute a célula abaixo para importar as bilbiotecas necessárias.

[3]:
```
import tensorflow as tf
print("Using TensorFlow Version:", tf.__version__)

import tensorflow_datasets as tfds
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
```

```
Using TensorFlow Version: 2.4.1
```

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.

2.1 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.

[4]:
```
# Carrega dados do TFDS
# Inclua seu código aqui
train_data, info = tfds.load('eurosat/rgb', split='train[:80%]', with_info=True,␣
 ↪as_supervised=True)
val_data =tfds.load('eurosat/rgb', split='train[80%:]', as_supervised=True)
```

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

[5]:
```
info
```

[5]:
```
tfds.core.DatasetInfo(
    name='eurosat',
    version=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',
    homepage='https://github.com/phelber/eurosat',
    features=FeaturesDict({
        'filename': Text(shape=(), dtype=tf.string),
        'image': Image(shape=(64, 64, 3), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=27000,
    splits={
        'train': 27000,
    },
    supervised_keys=('image', 'label'),
    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}
    }""",
    redistribution_info=,
)
```

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

[6]:
```
print('Número exemplos de treinamento =', len(list(train_data)))
print('Número exemplos de treinamento =', len(list(val_data)))
```

```
Número exemplos de treinamento = 21600
Número exemplos de treinamento = 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 conjutno de dados.

[7]:
```
labels_list = ['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway',␣
 ↪'Industrial',
               'Pasture', 'PermanentCrop', 'Residential', 'River', 'SeaLake']
print(labels_list)
```

```
['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway', 'Industrial',
'Pasture', 'PermanentCrop', 'Residential', 'River', 'SeaLake']
```

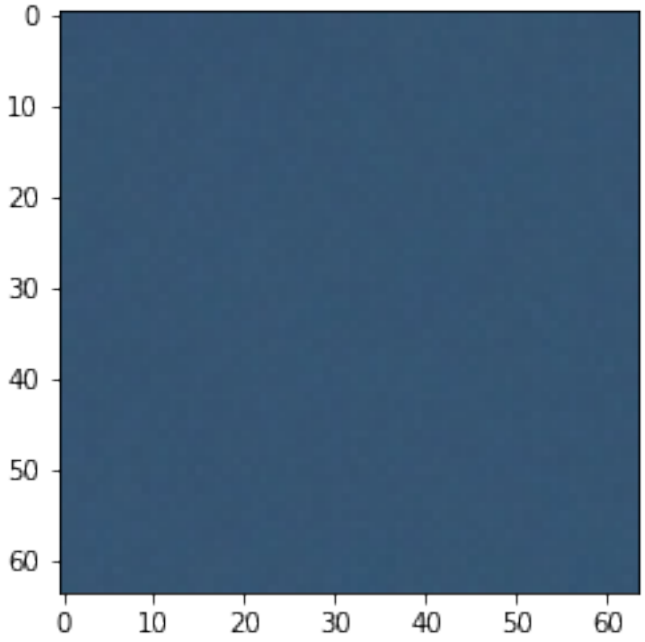
2.2 Exercíco #2: Visualização dos dados

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

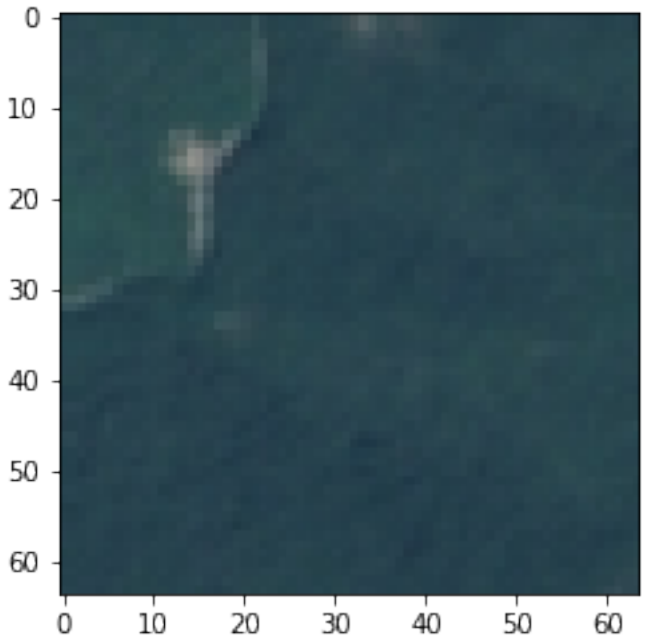
Para fazer isso você vai precisar de um laço `for`, dos comandos `print`, `plt.imshow()`, `plt.show()`, e do método `take()`.

[8]:
```
# Itera no conjunto de dados pegando exemplos
for data in train_data.take(5):
    image, label = data

    print("Classe: {} - {}".format(labels_list[label], label))
    plt.imshow(image)
    plt.show()

# Inclua seu código aqui
```

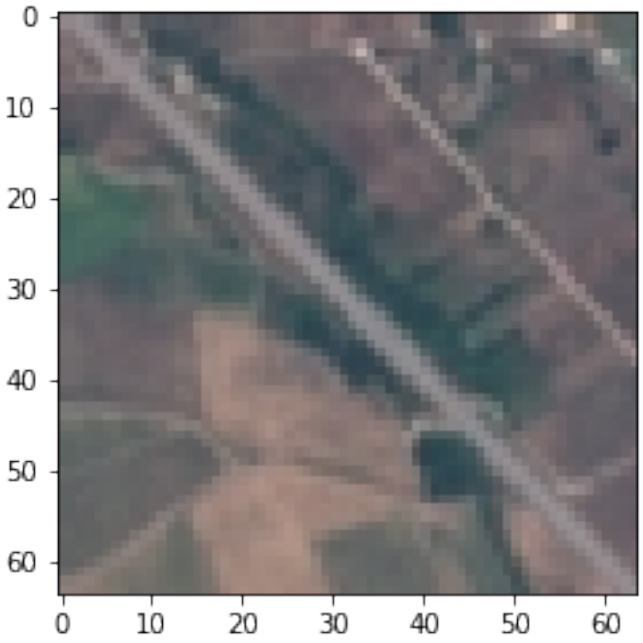
```
Classe: SeaLake - 9
```



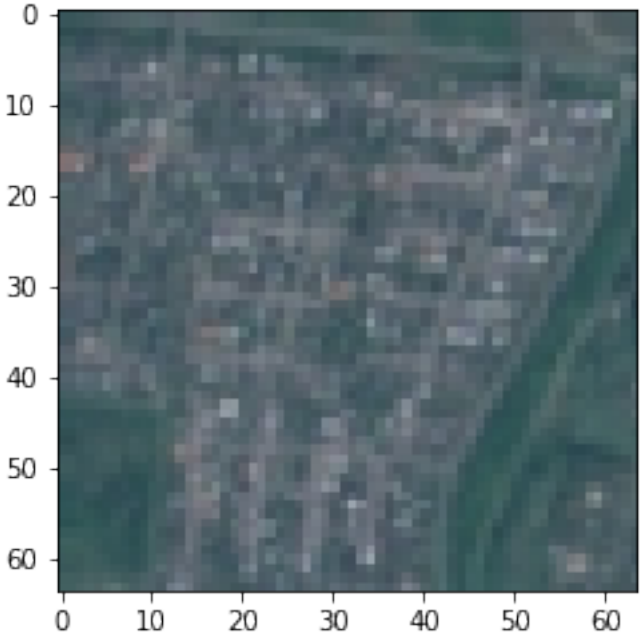

```
Classe: Forest - 1
```



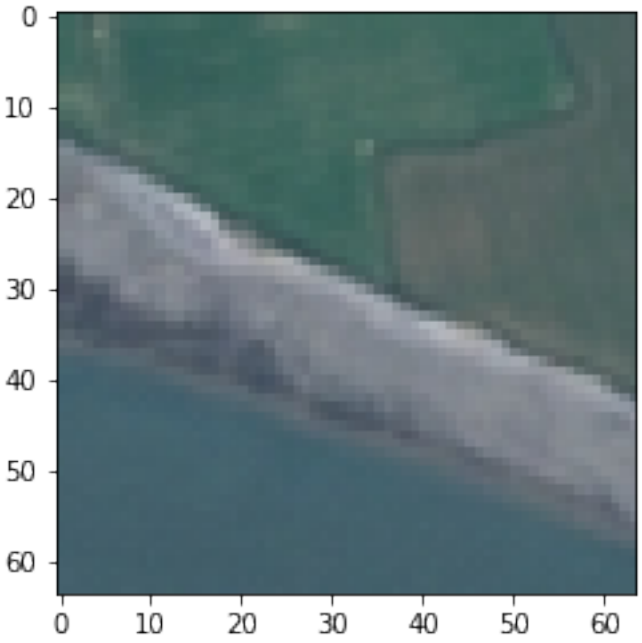

```
Classe: Highway - 3
```




```
Classe: Residential - 7
```




```
Classe: River - 8
```



Exemplo de saída esperada (sem as figuras):

```
Classe: Industrial - 4
Classe: River - 8
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.

3.1 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. A normalização deve transformar os valores dos pixels das imagens para números reais entre 0 e 1.

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 as imagens crie uma função de nome `format_image` e a utilize chamando o método `map()`. Para normalizar os dados você vai precisar primeiramente transformá-los em

`float32`, para isso use a função do TensorFlow `tf.cast()`. As instruções de uso dessa função podem ser vistas em https://www.tensorflow.org/api_docs/python/tf/cast.

[14]:
```
# Definição da dimensão das imagens para processamento e do tamanho dos lotes de␣
 ↪dados
# Inclua seu código aqui
IMAGE_SIZE = (64, 64)
BATCH_SIZE = 32

# Função usada para redimensionar e normalizar as imagens
def format_image(image, label):
    image_f32 = tf.cast(image, tf.float32)
    img = tf.image.resize(image_f32, IMAGE_SIZE) / 255.
    # Inclua seu código aqui )
    return img, label

# Cria lotes de dados usando o método map() para chamar a função format_image()
# Inclua seu código aqui
train_batches = train_data.map(format_image).batch(BATCH_SIZE)
val_batches = val_data.map(format_image).batch(BATCH_SIZE)

train_batches
```

[14]:
```
<BatchDataset shapes: ((None, 64, 64, 3), (None,)), types: (tf.float32,
tf.int64)>
```

Saída espeada:

```
<DatasetV1Adapter shapes: ((None, 64, 64, 3), (None,)), types: (tf.float32, tf.int64)>
```

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.

4.1 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()`.

[15]:
```
# Dimensão das imagens para argumento input_shape
# Inclua seu código aqui
IMAGE_DIM = (64, 64, 3)

# Carrega vetores de características com a URL do módulo
# Inclua seu código aqui
MODULE_HANDLE = "https://tfhub.dev/tensorflow/efficientnet/b0/feature-vector/1"
feature_extractor = hub.KerasLayer(MODULE_HANDLE, trainable=False,␣
 ↪input_shape=IMAGE_DIM)
feature_extractor
```

[15]:
```
<tensorflow_hub.keras_layer.KerasLayer at 0x7fd641dfc390>
```

Saída espeada:

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

4.2 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.

[21]:
```
# Número de classes da RNA
# Inclua seu código aqui
NUM_CLASSES = len(labels_list)

# Cria modelo seguencial do Keras para problema de classificação com 10 classes
# Inclua seu código aqui
rna = tf.keras.models.Sequential([
    feature_extractor,
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')]
)


# Apresenta configuração da RNA
# Inclua seu código aqui
rna.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1280)              4049564
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 10)                12810
=================================================================
Total params: 4,062,374
Trainable params: 12,810
Non-trainable params: 4,049,564
_________________________________________________________________
```

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.

5.1 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`.

[22]:
```
# Congela parâmetros da EfficientNet
# Inclua seu código aqui
feature_extractor.trainable = False

# Define método de otimização
# Inclua seu código aqui

# Compila RNA
# Inclua seu código aqui
```

```
rna.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics='accuracy')
```

5.2 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 10 épocas de treinamento.

[23]:
```
# Define número de épocas de treinamento
# Inclua seu código aqui
TRAINING_EPOCHS = 10

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

```
Epoch 1/10
675/675 [==============================] - 16s 21ms/step - loss: 0.7933 -
accuracy: 0.7713 - val_loss: 0.3186 - val_accuracy: 0.8980
Epoch 2/10
675/675 [==============================] - 14s 20ms/step - loss: 0.2819 -
accuracy: 0.9081 - val_loss: 0.2703 - val_accuracy: 0.9113
Epoch 3/10
675/675 [==============================] - 14s 20ms/step - loss: 0.2340 -
accuracy: 0.9230 - val_loss: 0.2500 - val_accuracy: 0.9176
Epoch 4/10
675/675 [==============================] - 13s 20ms/step - loss: 0.2073 -
accuracy: 0.9335 - val_loss: 0.2387 - val_accuracy: 0.9217
Epoch 5/10
675/675 [==============================] - 13s 20ms/step - loss: 0.1887 -
accuracy: 0.9408 - val_loss: 0.2318 - val_accuracy: 0.9244
Epoch 6/10
675/675 [==============================] - 13s 20ms/step - loss: 0.1744 -
accuracy: 0.9450 - val_loss: 0.2274 - val_accuracy: 0.9265
Epoch 7/10
675/675 [==============================] - 13s 20ms/step - loss: 0.1629 -
accuracy: 0.9488 - val_loss: 0.2245 - val_accuracy: 0.9272
Epoch 8/10
675/675 [==============================] - 13s 20ms/step - loss: 0.1532 -
accuracy: 0.9523 - val_loss: 0.2227 - val_accuracy: 0.9270
Epoch 9/10
675/675 [==============================] - 13s 20ms/step - loss: 0.1450 -
accuracy: 0.9544 - val_loss: 0.2216 - val_accuracy: 0.9287
Epoch 10/10
```

```
675/675 [==============================] - 13s 20ms/step - loss: 0.1378 -
accuracy: 0.9572 - val_loss: 0.2211 - val_accuracy: 0.9287
```

Saída esperada:

```
Epoch 1/10
675/675 [==============================] - 13s 19ms/step - loss: 0.5030 - accuracy: 0.8542 - val
.
.
.
Epoch 10/10
675/675 [==============================] - 12s 18ms/step - loss: 0.1355 - accuracy: 0.9574 - val
```

5.3 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:

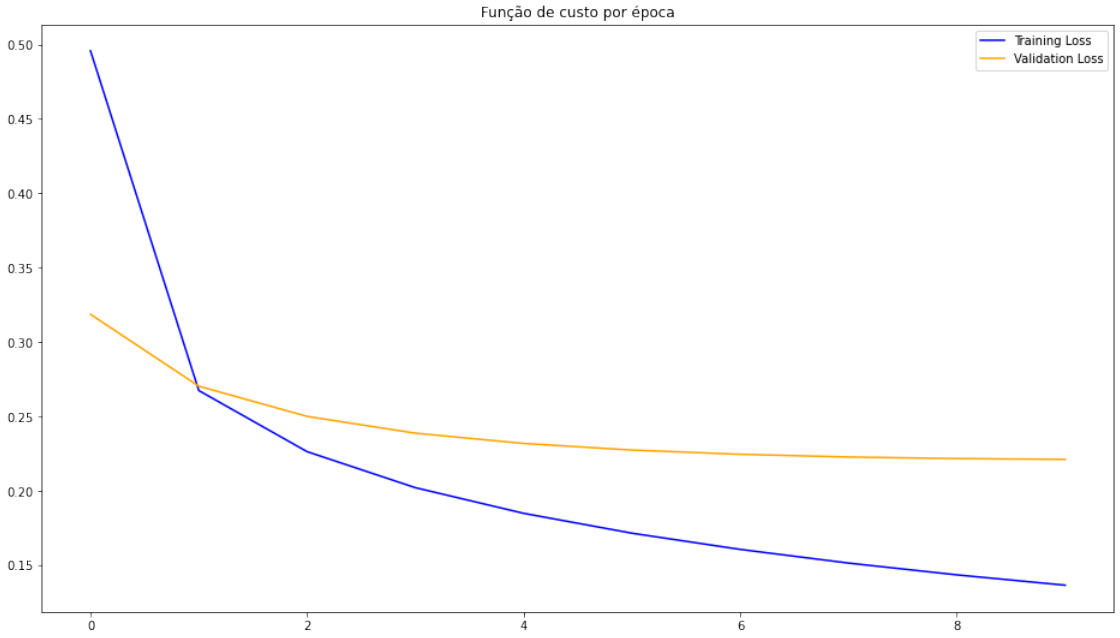
1. Valores da função de custo para os dados de treinamento e de validação;
2. Valores da métrica para os dados de treinamento e de validação.

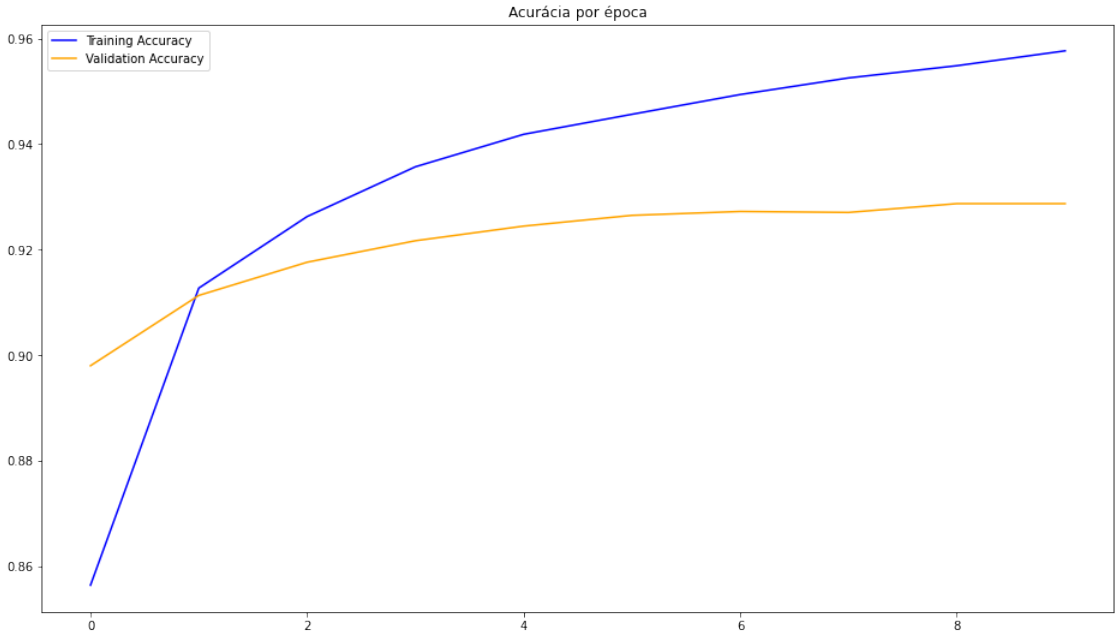
[27]:
```
history.history.keys()
```

[27]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[32]:
```
# Definir vetores com valores da função de custo e da métrica para os dados de␣
 ↪treinamento e de validação
# Inclua seu código aqui
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (16,9)

# Criar vetor de épocas
# Inclua seu código aqui
eps = np.arange(len(history.history['loss']))

# Fazer o gráfico dos valores da função de custo
# Inclua seu código aqui
plt.plot(eps, history.history['loss'], label='Training Loss', c='blue')
plt.plot(eps, history.history['val_loss'], label='Validation Loss', c='orange')
plt.legend()
plt.title("Função de custo por época")
plt.show()

# Fazer o gráfico dos valores da métrica
# Inclua seu código aqui
plt.plot(eps, history.history['accuracy'], label='Training Accuracy', c='blue')
plt.plot(eps, history.history['val_accuracy'], label='Validation Accuracy',␣
 ↪c='orange')
plt.legend()
```

```
plt.title("Acurácia por época")
plt.show()
```





5.4 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`.

5.5 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 de custo e da métrica para os dados de validação.

[35]:
```
# Avalia desempenho da RNA para os dados de validação
# Inclua seu código aqui
val_loss, val_accuracy = rna.evaluate(val_batches)

# Apresenta os resultados
# Inclua seu código aqui
print(f'loss: {round(val_loss, 4)}\naccuracy: {round(val_accuracy, 4)}')
```

```
169/169 [==============================] - 3s 16ms/step - loss: 0.2211 -
accuracy: 0.9287
loss: 0.2211
accuracy: 0.9287
```

Saída esperada:

```
loss: 0.2189
accuracy: 0.9244
```

5.6 Exercício #10: Teste de classificação de imagens

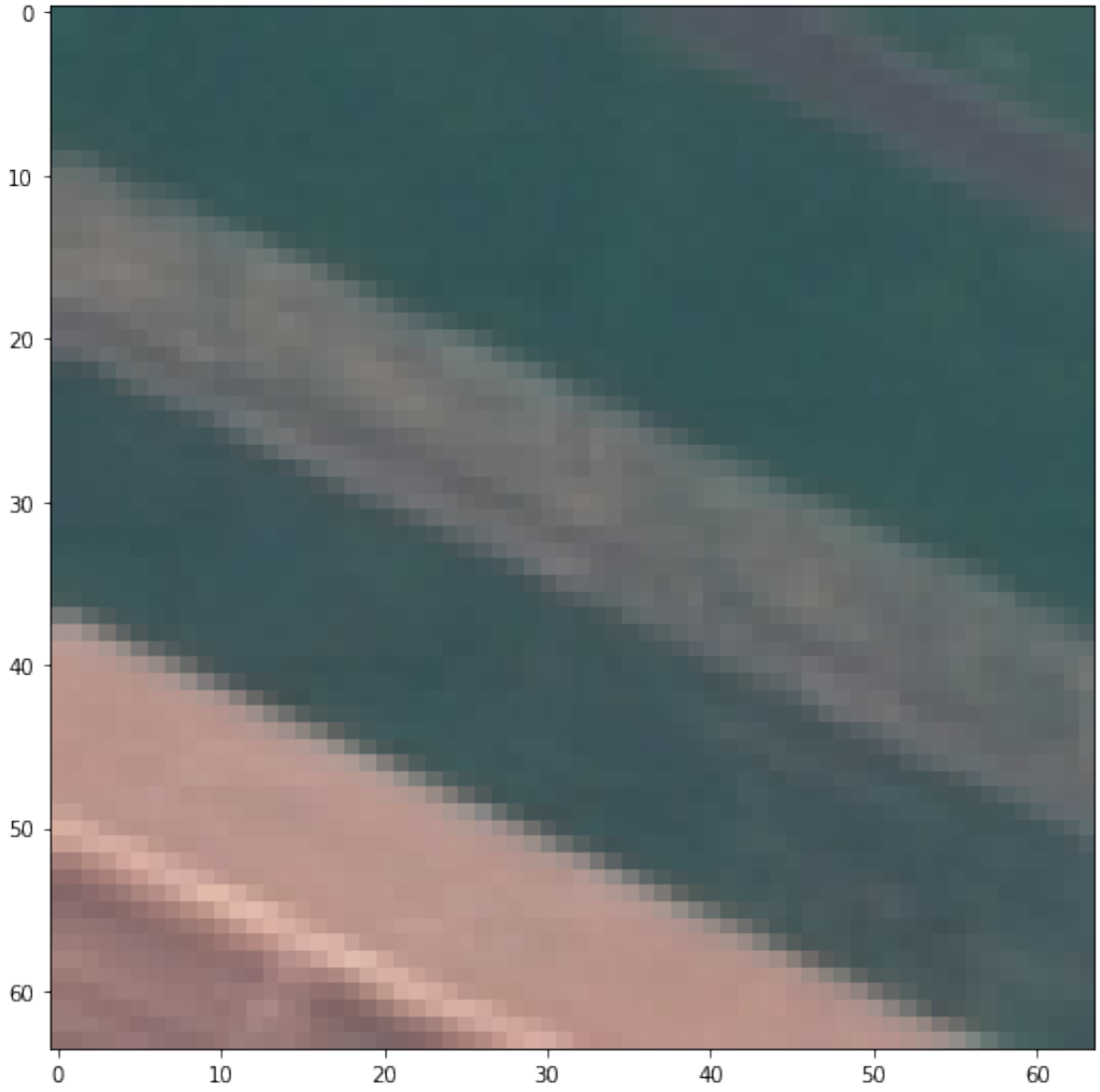
Para poder fornecer as imagens para a RNA usando o método `predict` você precisa extrai-las do objeto `val_data` e processá-las com a função `format_image`, que por sua vez é chamada pelo método `map()`. Além disso, você tem que incluir o eixo dos exemplos na imagem de acordo com o esperado por uma RNA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

[41]:
```
# Itera no objeto val_data para pegar 5 imagens e aplica função format_image
# Inclua seu código aqui (~1 linha)
for data in val_data.map(format_image).take(5):

    # Extrai imagem e classe prevista
    # Inclua seu código aqui
    image, label = data
    # Adiciona eixo dos exemplos
```

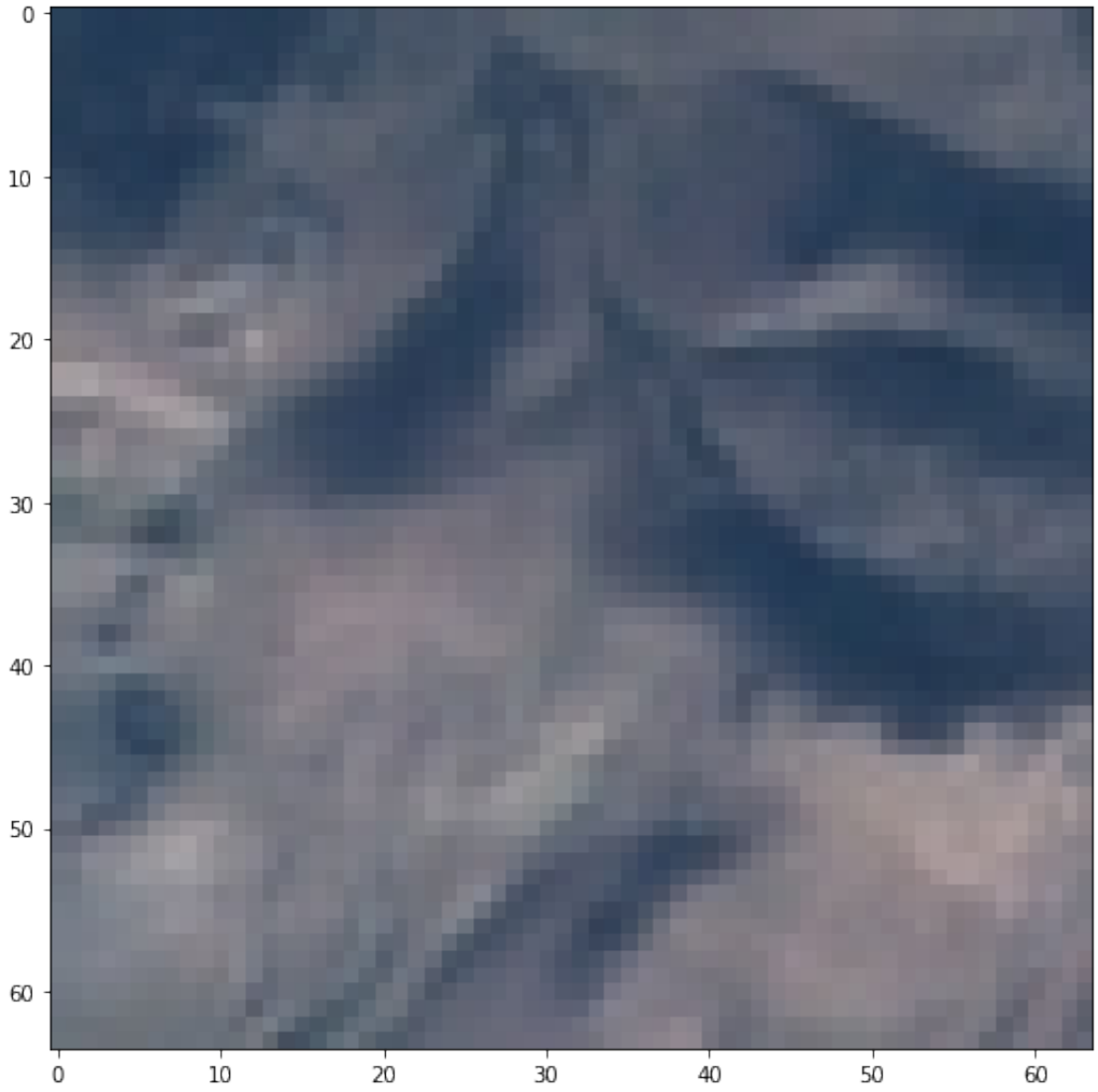
```
    # Inclua seu código aqui
    image = np.expand_dims(image, axis=0)

    # Calcula probabilidades previstas pela RNA
    # Inclua seu código aqui
    img_pred = np.argmax(rna.predict(image))
    # Determina classe prevista
    # Inclua seu código aqui
    print('Classe prevista =', labels_list[img_pred], ', Classe real =',␣
↪labels_list[label.numpy()])
    plt.imshow(image[0])
    plt.show()


    # Apresenta resultados das classes e mostra imagem
    # Inclua seu código aqui
```

```
Classe prevista = AnnualCrop , Classe real = AnnualCrop
```

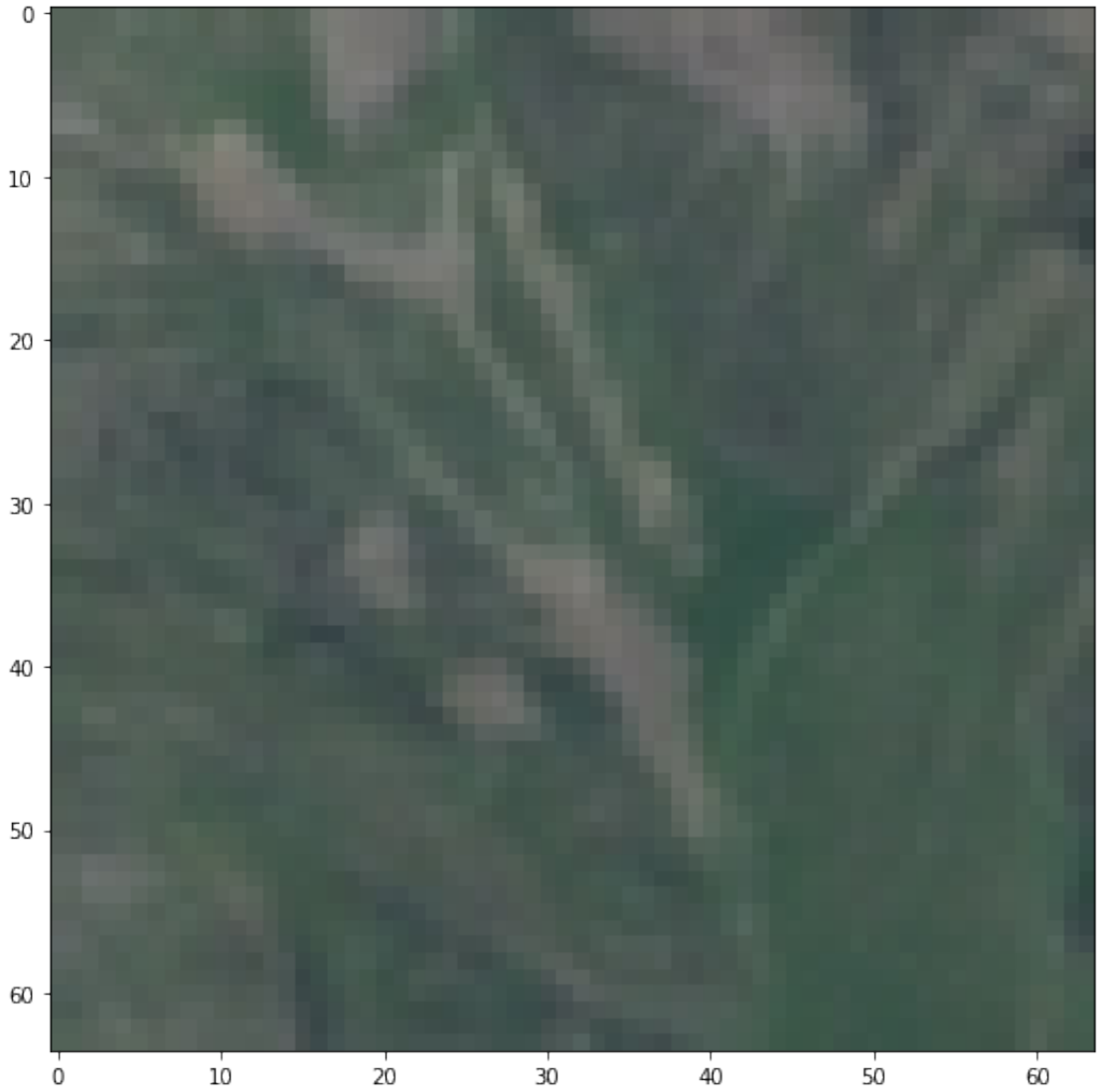

0
10
20
30
40
50
60
0
10
20
30
40
50
60

```
Classe prevista = Pasture , Classe real = Pasture
```

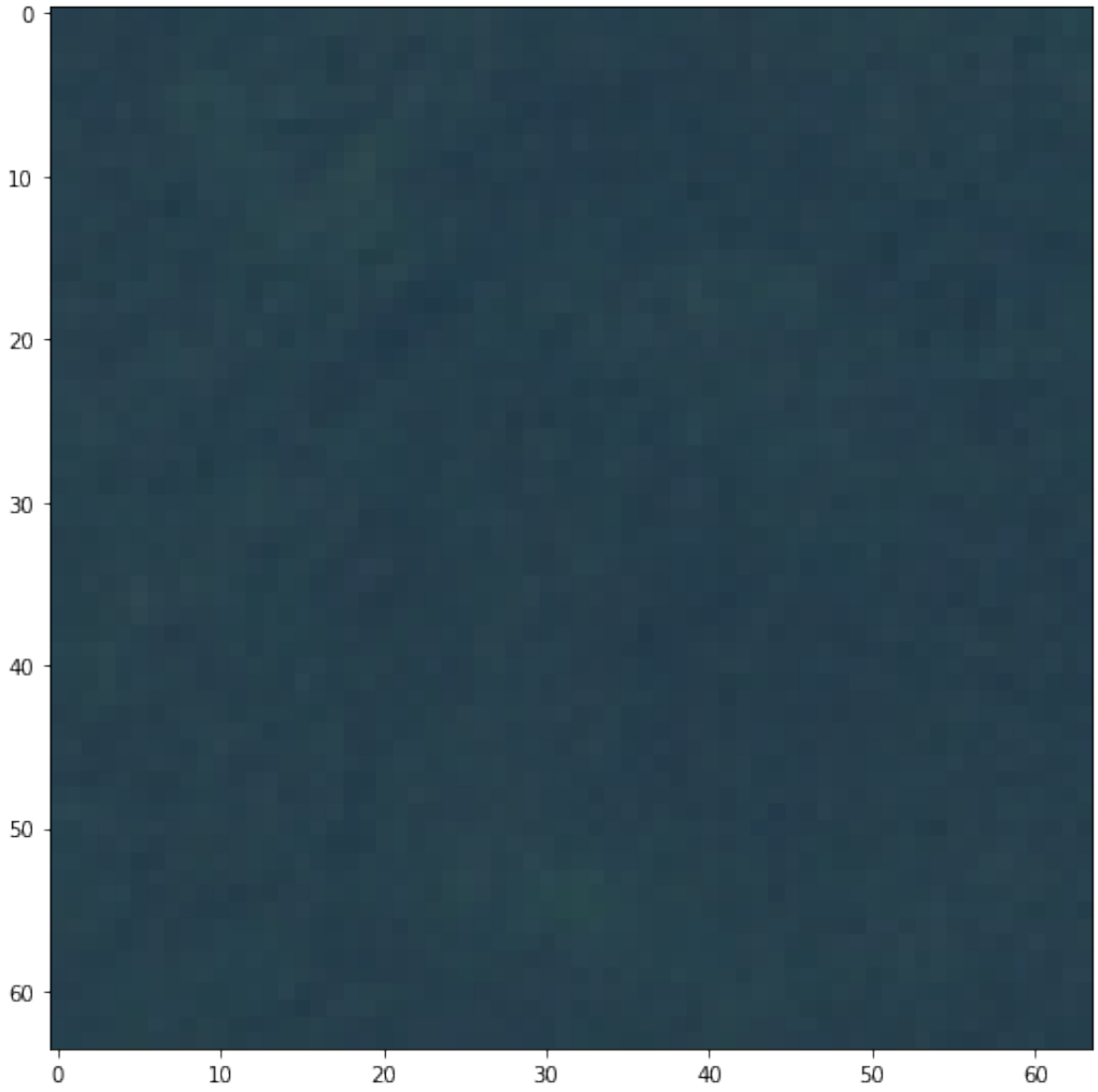
```
Classe prevista = HerbaceousVegetation , Classe real = HerbaceousVegetation
```



```
Classe prevista = HerbaceousVegetation , Classe real = HerbaceousVegetation
```



```
Classe prevista = Forest , Classe real = Forest
```



Saída esperada (sem incluir as imagens):

```
Classe prevista = Industrial , Classe real = Industrial
Classe prevista = AnnualCrop , Classe real = AnnualCrop
Classe prevista = HerbaceousVegetation , Classe real = HerbaceousVegetation
Classe prevista = River , Classe real = River
Classe prevista = Residential , Classe real = Residential
```