Aula 10

Redes Complexas

Eduardo Lobo Lustosa Cabral

1. Objetivo

Aprsentar a classe Funcional do Keras.

Apresentar como criar algumas redes complexas no Keras.

Carregar bibliotecas principais

Em primeiro lugar é necessário importar alguns pacotes do Python que serão usados:

- Numpy pacote de cálculo científico com Python
- Matplotlib biblioteca para gerar gráficos em Python
- TensorFlow

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

2. Classe Funcional do Keras

Modelo sequencial do Keras não serve para RNAs que possuem mais do que uma entrada ou uma saída e também para RNAs mais complexas que possuem laços ou estruturas em paralelo.

Para configurar uma RNA com fluxo de informação não sequencial tem que usar a Classe Funcional do Keras ("Functional API").

A classe Funcional do Keras permite construir RNAs com vários ramos, várias entradas, várias saídas etc.

A classe Funcional do Keras é muito flexível e permite construir praticamente qualquer tipo de RNA.

Exemplo de uma RNA configurada com a classe de modelo funcional do Keras

```
# Importa classe funcional
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Configuração da RNA
X = Input(shape=(32,))
```

```
A = Dense(64, activation='relu')(X)
Y = Dense(16, activation='softmax')(A)

# Criação da RNA
rna = Model(inputs=X, outputs=Y)

rna.summary()
```

```
Model: "functional"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)             │ (None, 32)                  │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense (Dense)                        │ (None, 64)                  │           2,112 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 16)                  │           1,040 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
 Total params: 3,152 (12.31 KB)

 Trainable params: 3,152 (12.31 KB)

 Non-trainable params: 0 (0.00 B)
```

- Essa RNA inclui todas as camadas necessárias para o cálculo de Y dado X.

Comparação entre uma RNA criada com a classe sequencial e a mesma RNA cria com a classe funcional

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras import layers

# RNA criada com classe sequencial da forma que conhecemos
rna_seq = Sequential()
rna_seq.add(layers.Dense(128, activation='relu',input_shape=(64,)))
rna_seq.add(layers.Dense(32, activation='relu'))
rna_seq.add(layers.Dense(10, activation='softmax'))
rna_seq.summary()
print('\n')
```

```
# Mesma RNA criada com classe funcional
x0 = layers.Input(shape=(64,))
x = layers.Dense(128, activation='relu', name='teste')(x0)
x = layers.Dense(32, activation='relu')(x)
output = layers.Dense(10, activation='softmax')(x)
rna = Model(x0, output)
rna.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense_2 (Dense)                      │ (None, 128)                 │
8,320 │
├──────────────────────────────────────┼─────────────────────────────┼─
──────────────┤
│ dense_3 (Dense)                      │ (None, 32)                  │
4,128 │
├──────────────────────────────────────┼─────────────────────────────┼─
──────────────┤
│ dense_4 (Dense)                      │ (None, 10)                  │
330 │
└──────────────────────────────────────┴─────────────────────────────┴─
──────────────┘
```

```
 Total params: 12,778 (49.91 KB)

 Trainable params: 12,778 (49.91 KB)

 Non-trainable params: 0 (0.00 B)
```

```
Model: "functional_4"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
```

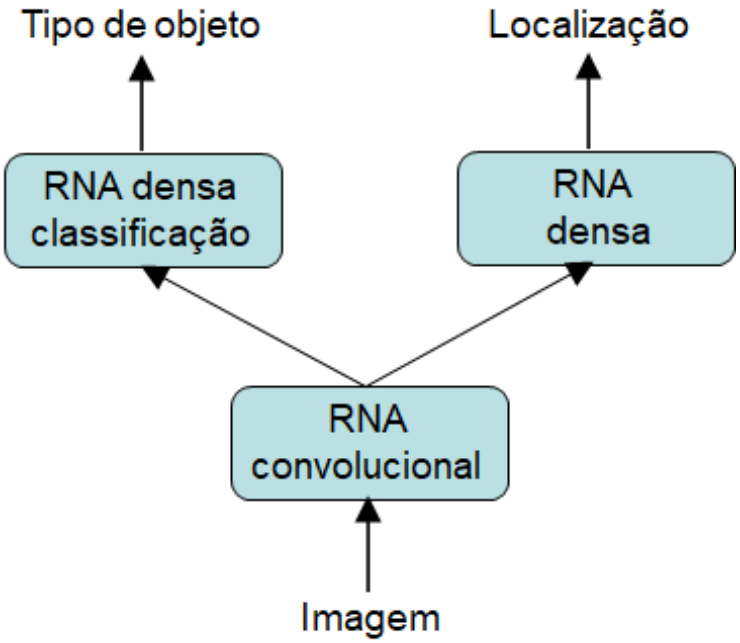
```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳
━━━━━━━━━━━━━━━━━┓
┃ input_layer_2 (InputLayer)            ┃ (None, 64)                   ┃
0 ┃
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ teste (Dense)                         │ (None, 128)                  │
8,320 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_5 (Dense)                       │ (None, 32)                   │
4,128 │
├───────────────────────────────────────┼──────────────────────────────┼
─────────────────┤
│ dense_6 (Dense)                       │ (None, 10)                   │
330 │
└───────────────────────────────────────┴──────────────────────────────┴
─────────────────┘

 Total params: 12,778 (49.91 KB)

 Trainable params: 12,778 (49.91 KB)

 Non-trainable params: 0 (0.00 B)
```

3. Redes complexas

Muitas aplicações exigem RNAs mais complexas do que as que já estudamos, como por exemplo:

- Detecção e localização de objetos em imagens → nesse caso a RNA tem que ter várias saídas de tipos diferentes;
- RNAs que processam várias entradas em paralelo com a mesma RNA (visão estéreo e one-shot learning) → nesse caso a RNA tem dois ramos iguais (RNA Siamesa);
- Transferência de estilo → nesse caso tem-se duas imagens que são processadas em paralelo por duas RNAs iguais e as saídas são comparadas para gerar o erro de treinamento.
- Segmentação de imagens, autoencoders e GANs → nesses casos a rede tem que ser capaz de gerar imagens a partir de vetores com dimensão muito menor ou criar imagens de dimensão maior do que a imagem recebida como dado de entrada.

3.1 RNA com várias saídas



(rna_complexa_2.png)

Exemplo de configuração no Keras

- Parte convolucional:
  - Uma camada convolucional com 128 filtros de dimensão 3x3 e função de ativação Relu
  - Uma camada Maxpooling
  - Uma camada convolucional com 256 filtros de dimensão 3x3 e função de ativação Relu
  - Uma camada Maxpooling
- Parte densa para classificação:
  - Uma camada tipo densa com 64 neurônios e função de ativação Relu
  - Uma camada de saída com 10 neurônios (10 classes) e função de ativação softmax
- Parte densa para localização do objeto:
  - Uma camada tipo densa com 64 neurônios e função de ativação Relu
  - Uma camada de saída com 4 neurônios e função de ativação sigmoide

```
from tensorflow.keras import layers
from tensorflow.keras.models import Model

# Definição da entrada
input_shape = (64, 64, 3)
X = layers.Input(shape=input_shape)

# Configuração da parte convoluciona
X1 = layers.Conv2D(128, (3,3), activation='relu')(X)
```

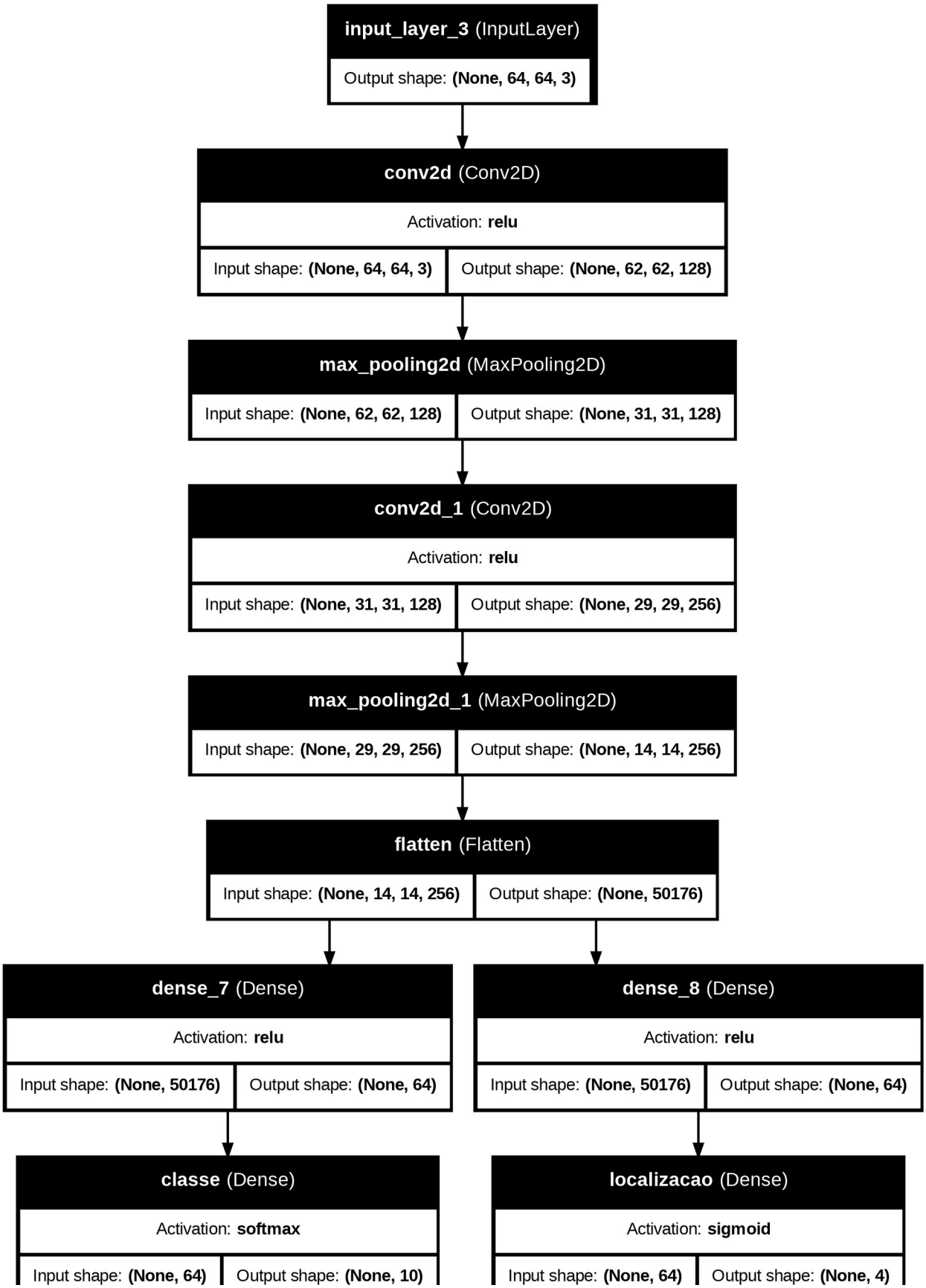
```
X1 = layers.MaxPooling2D(2, 2)(X1)
X2 = layers.Conv2D(256, (3,3), activation='relu')(X1)
X3 = layers.MaxPooling2D(2, 2)(X2)

# Camada de Flatten
X4 = layers.Flatten()(X3)

# Rede de classificação para 10 classes
X5 = layers.Dense(64, activation='relu')(X4)
Y_class = layers.Dense(10, activation='softmax', name='classe')(X5)

# Rede de localização
X6 = layers.Dense(64, activation='relu')(X4)
Y_loc = layers.Dense(4, activation='sigmoid', name='localizacao')(X6)

# Criação da RNA
rna = Model(inputs=X, outputs=[Y_class, Y_loc])

# Mostra resumo da RNA
rna.summary()
```

```
Model: "functional_5"
```

Layer (type) Connected to	Output Shape	Param #
input_layer_3 - (InputLayer)	(None, 64, 64, 3)	0
conv2d (Conv2D) input_layer_3[0][0]	(None, 62, 62, 128)	3,584
max_pooling2d conv2d[0][0] (MaxPooling2D)	(None, 31, 31, 128)	0
conv2d_1 (Conv2D) max_pooling2d[0][0]	(None, 29, 29, 256)	295,168
max_pooling2d_1	(None, 14, 14, 256)	0

```
│ conv2d_1[0][0]          │
│ (MaxPooling2D)          │                        │
│                         │
├─────────────────────────┼────────────────────────┼──────────────
─┼─────────────────────────┤
│ flatten (Flatten)       │ (None, 50176)          │             0
│ max_pooling2d_1[0][0]   │
├─────────────────────────┼────────────────────────┼──────────────
─┼─────────────────────────┤
│ dense_7 (Dense)         │ (None, 64)             │     3,211,328
│ flatten[0][0]           │
├─────────────────────────┼────────────────────────┼──────────────
─┼─────────────────────────┤
│ dense_8 (Dense)         │ (None, 64)             │     3,211,328
│ flatten[0][0]           │
├─────────────────────────┼────────────────────────┼──────────────
─┼─────────────────────────┤
│ classe (Dense)          │ (None, 10)             │           650
│ dense_7[0][0]           │
├─────────────────────────┼────────────────────────┼──────────────
─┼─────────────────────────┤
│ localizacao (Dense)     │ (None, 4)              │           260
│ dense_8[0][0]           │
└─────────────────────────┴────────────────────────┴──────────────
─┴─────────────────────────┘

 Total params: 6,722,318 (25.64 MB)

 Trainable params: 6,722,318 (25.64 MB)

 Non-trainable params: 0 (0.00 B)
```

```python
# Cria um gráfico da RNA no arquivo rna.png
from tensorflow.keras.utils import plot_model
plot_model(rna, to_file='rna_detect.png', show_shapes=True,
show_layer_activations=True, show_layer_names=True)
```


input_layer_3 (InputLayer)
Output shape: (None, 64, 64, 3)
conv2d (Conv2D)
Activation: relu
Input shape: (None, 64, 64, 3)
Output shape: (None, 62, 62, 128)
max_pooling2d (MaxPooling2D)
Input shape: (None, 62, 62, 128)
Output shape: (None, 31, 31, 128)
conv2d_1 (Conv2D)
Activation: relu
Input shape: (None, 31, 31, 128)
Output shape: (None, 29, 29, 256)
max_pooling2d_1 (MaxPooling2D)
Input shape: (None, 29, 29, 256)
Output shape: (None, 14, 14, 256)
flatten (Flatten)
Input shape: (None, 14, 14, 256)
Output shape: (None, 50176)
dense_7 (Dense)
Activation: relu
Input shape: (None, 50176)
Output shape: (None, 64)
dense_8 (Dense)
Activation: relu
Input shape: (None, 50176)
Output shape: (None, 64)
classe (Dense)
Activation: softmax
Input shape: (None, 64)
Output shape: (None, 10)
localizacao (Dense)
Activation: sigmoid
Input shape: (None, 64)
Output shape: (None, 4)

Compilação de uma RNA com várias saídas

Para compilar um RNA com várias saídas deve-se especificar uma função de custo e uma métrica para cada saída. Além disso é necessário definir pesos para cada função de custo de forma a compor a função de custo global, que é utilizada no treinamento da rede.

A célula abaixo mostra um exemplo de como compilar a rede configurada anteriormente que possui uma saída com 10 neurônios para classificação multiclasse e outra saída com 4 neurônios para ajuste de um função. Para essas saídas são definidas:

- Saída de classificação: função de custo `categorical_crossentropy` e métrica `accuracy`
- Saída de ajuste de função: função de custo `mse` e métrica `mae`
- Pesos para as funções de custo: 2 para `localização (mse)` e 1 para `classificação (categorical_crossentropy)`

```
# importa do keras a classe dos otimizadores
from tensorflow.keras import optimizers

# Configuração do otimizador
adam = optimizers.Adam(learning_rate=0.001)

rna.compile(optimizer=adam,
            loss={
                  'classe': 'categorical_crossentropy',
                  'localizacao': 'mse'},
            loss_weights={
                  'localizacao': 2.0,
                  'classe': 1.0},
            metrics={
                  'classe': 'accuracy',
                  'localizacao': 'mae'})
```

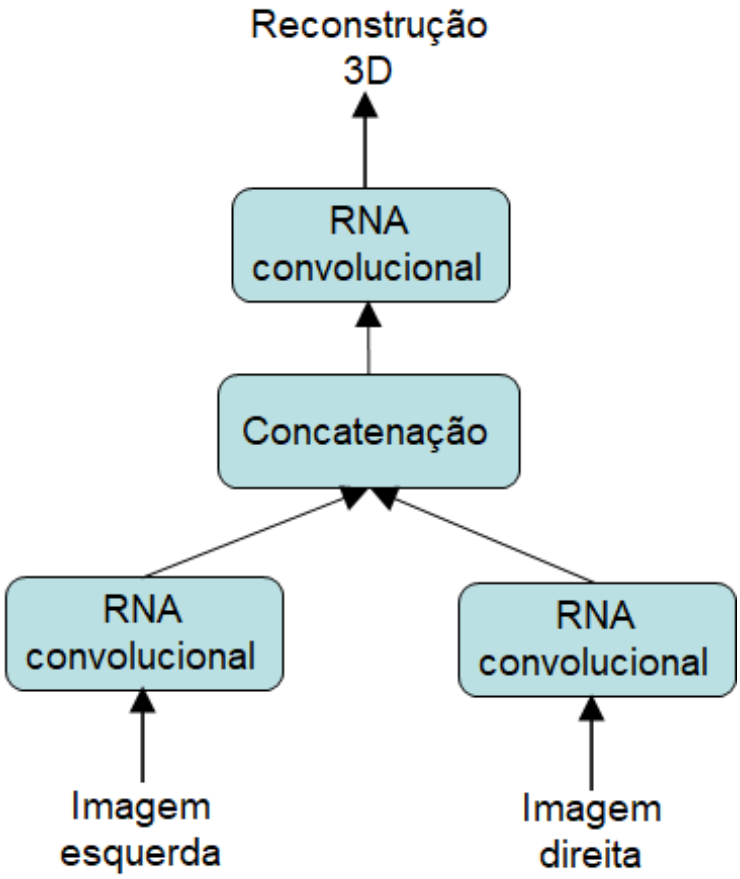
- Observa-se que os nomes `classe` e `localizacao` usados na compilação, são os nomes definidos para as duas camadas de saída dessa rede.

Treinamento da RNA

O treinamento de uma RNA com várias saídas usando o método `fit` é realizado da mesma forma que para qualquer outra rede. A única diferença é que deve-se passar todas as saídas ao executar o método `fit`. A célula baixo mostra como treinar essa RNA com duas saídas.

```
history = rna.fit(x_train, [y_class_train, y_loc_train], epochs=100,
verbose=2, validation_data=(x_val, [y_class_val, y_loc_val]))
```

3.2 RNA com mais de um ramo (siamesa)



(rna_complexa_3.png)

Exemplo de configuração no Keras

- Rede convolucional siamesa:
  - Uma camada convolucionial com 128 filtros de dimensão 3x3 e função de ativação Relu
  - Uma camada convolucionial com 256 filtros de dimensão 3x3 e função de ativação Relu
- Parte convolucional de reconstrução 3D
  - Uma camada convolucionial com 128 filtros de dimensão 3x3, stride = (2,1) e função de ativação Relu
  - Uma camada convolucionial com 32 filtros de dimensão 3x3 e função de ativação Relu
  - Uma camada convolucionial com 1 filtro de dimensão 1x1 e função de ativação Relu

A camada de saída deve ter somente um filtro porque o que é desejado no caso são as distâncias dos objetos da cena em relação às cameras.

```
from tensorflow.keras import layers
from tensorflow.keras.models import Model, Sequential

# Configuração da rede convolucional siamesa
img_dim = (120, 160, 3)

rna_siam = Sequential()
rna_siam.add(layers.Conv2D(128, (3,3), activation='relu',
padding='same', input_shape=img_dim))
rna_siam.add(layers.Conv2D(256, (3,3), activation='relu',
padding='same'))

# Mostra resumo da RNA
rna_siam.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d_2 (Conv2D)                    │ (None, 120, 160, 128)       │
3,584 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 120, 160, 256)       │
295,168 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
 Total params: 298,752 (1.14 MB)

 Trainable params: 298,752 (1.14 MB)

 Non-trainable params: 0 (0.00 B)
```

```
# Define as entradas da rede
img_esq = layers.Input(shape=img_dim, name='img_esq')
img_dir = layers.Input(shape=img_dim, name='img_dir')

# Processa imagens para gerar imagens de características
```

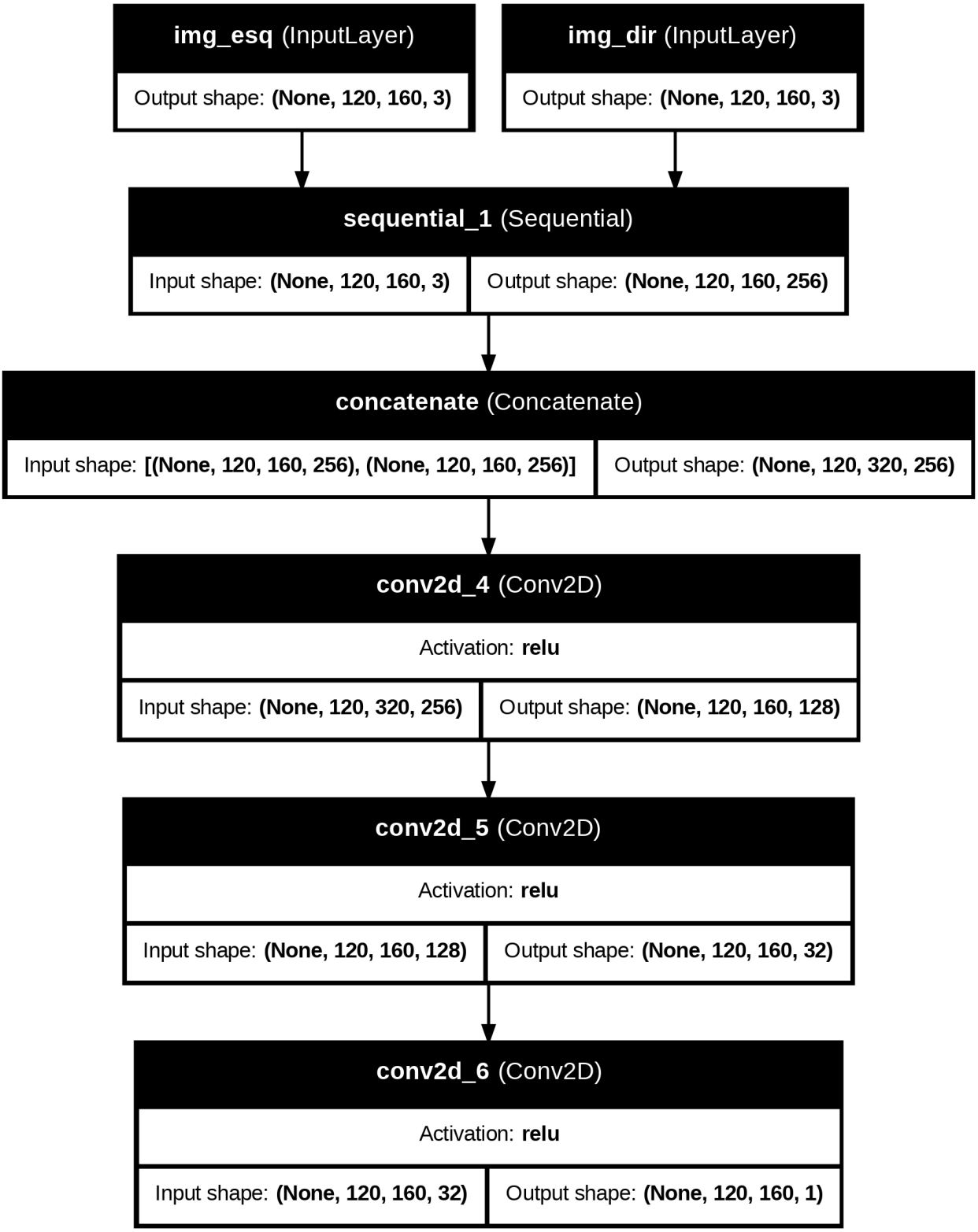
```
carac_esq = rna_siam(img_esq)
carac_dir = rna_siam(img_dir)

# Une as imagens de características
X_con = layers.concatenate([carac_esq, carac_dir], axis=2)

# Processa imagens de caracteríticas para gerar imagem de 3D
X1 = layers.Conv2D(128, (3,3), strides=(1,2), activation='relu',
padding='same')(X_con)
X2 = layers.Conv2D(32, (3,3), activation='relu', padding='same')(X1)
Y = layers.Conv2D(1, (1,1), activation='relu')(X2)

# cria rede siamesa
rna = Model(inputs=[img_esq, img_dir], outputs=Y)

# Mostra resumo da RNA
rna.summary()
```

```
Model: "functional_8"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)              ┃ Output Shape              ┃         Param # ┃
┃ Connected to              ┃                           ┃                 ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ img_esq (InputLayer)      │ (None, 120, 160, 3)       │               0 │
│ -                         │                           │                 │
├───────────────────────────┼───────────────────────────┼─────────────────┤
│ img_dir (InputLayer)      │ (None, 120, 160, 3)       │               0 │
│ -                         │                           │                 │
├───────────────────────────┼───────────────────────────┼─────────────────┤
│ sequential_1 (Sequential) │ (None, 120, 160, 256)     │         298,752 │
│ img_esq[0][0],            │                           │                 │
│ img_dir[0][0]             │                           │                 │
├───────────────────────────┼───────────────────────────┼─────────────────┤
│ concatenate (Concatenate) │ (None, 120, 320, 256)     │               0 │
│ sequential_1[0][0],       │                           │                 │
│ sequential_1[1][0]        │                           │                 │
├───────────────────────────┼───────────────────────────┼─────────────────┤
│ conv2d_4 (Conv2D)         │ (None, 120, 160, 128)     │         295,040 │
│ concatenate[0][0]         │                           │                 │
└───────────────────────────┴───────────────────────────┴─────────────────┘
```

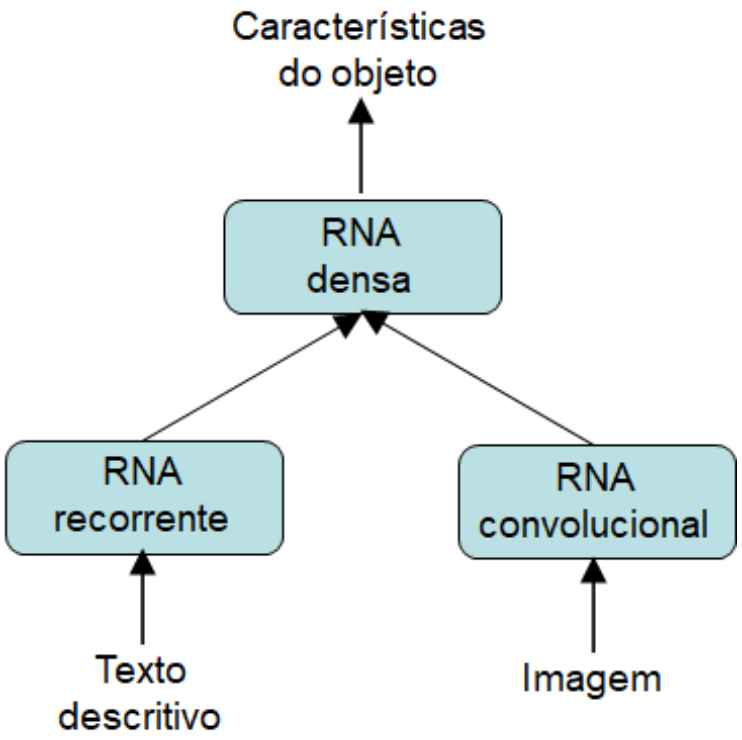
```
│ conv2d_5 (Conv2D)          │ (None, 120, 160, 32)   │          36,896
│ conv2d_4[0][0]             │                        │
├────────────────────────────┼────────────────────────┼─────────────────
┼────────────────────────────┤
│ conv2d_6 (Conv2D)          │ (None, 120, 160, 1)    │              33
│ conv2d_5[0][0]             │                        │
└────────────────────────────┴────────────────────────┴─────────────────
┴────────────────────────────┘

 Total params: 630,721 (2.41 MB)

 Trainable params: 630,721 (2.41 MB)

 Non-trainable params: 0 (0.00 B)
```

```python
# Cria um gráfico da RNA no arquivo rna.png
plot_model(rna, to_file='rna_siam.png', show_shapes=True,
show_layer_activations=True, show_layer_names=True)
```



3.3 RNA com diferentes tipos de entradas:

(rna_complexa_1.png)


Características
do objeto
RNA
densa
RNA
recorrente
RNA
convolucional
Texto
descritivo
Imagem