Aula 6
Redes complexas

Eduardo L. L. Cabral

Objetivos

- Apresentar RNAs de referência.
- Apresentar convolução 1x1.
- Apresentar RNAs complexas.
- Apresentar classe de modelos funcionais do Keras.

RNAs de referência

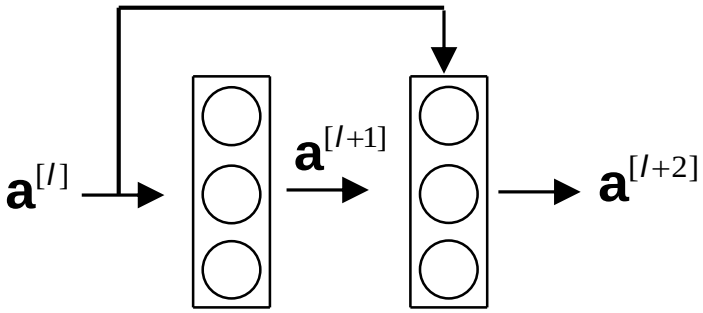
- Duas arquiteturas de RNAs convolucionais se tornaram referência na área de visão computacional pelas suas ideias inovadoras e desempenhos extraordinários:
  - RNAs residuais (“Residual networks”);
  - “Inception Networks”.
- Essas RNAs apresentam duas grandes inovações:
  - Caminhos alternativos do fluxo de informações (ResNet);
  - Convolução 1x1 (Inception).
- Essas são as RNAs com maior número de camadas que existem.

RNAs de referência - ResNet

- Referência ⇒ He et al., Deep residual networks for image recognition, 2015.
- RNAs muito profundas são difíceis de treinar pelo fato do gradiente diminuir a cada camada da rede durante o processo de propagação para trás.
- A ResNet possui um bloco denominado Bloco Residual (“Residual Block”) que consiste de um atalho entre duas camadas da rede não vizinhas.
- O bloco residual permite treinar de forma mais fácil RNAs muito profundas.

RNAs de referência - ResNet

- Esquema de um bloco residual:
  - As ativações da *l*-ésima camada, $\mathbf{a}^{[l]}$, servem como entradas da camada *l*+1 e também da camada *l*+2;
  - As ativações $\mathbf{a}^{[l]}$ se somam às ativações $\mathbf{a}^{[l+1]}$ e ambas se tornam entradas da camada *l*+2.

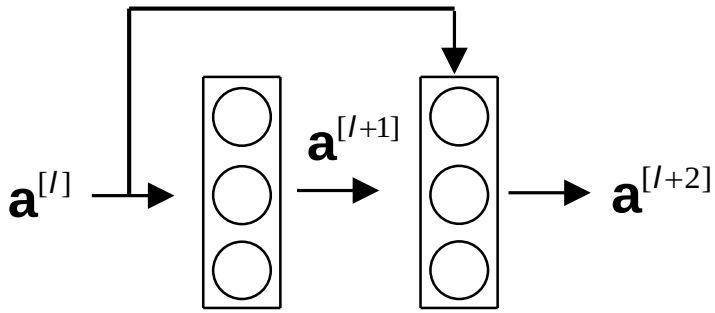


  - Nesse bloco existe o caminho padrão e o atalho que é somado ao caminho padrão.

RNAs de referência - ResNet

- Equações do bloco residual:

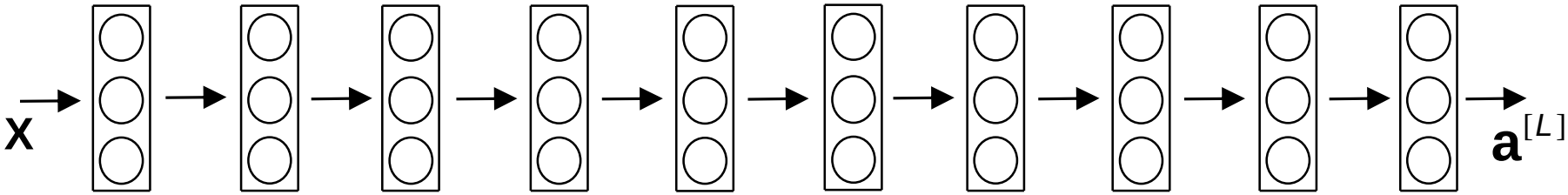
$$\begin{cases} \mathbf{z}^{[l+1]} = \mathbf{W}^{[l+1]}\mathbf{a}^{[l]} + \mathbf{b}^{[l+1]} \\ \mathbf{a}^{[l+1]} = g^{[l+1]}\left(\mathbf{z}^{[l+1]}\right) \\ \mathbf{z}^{[l+2]} = \mathbf{W}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} \\ \mathbf{a}^{[l+2]} = g^{[l+2]}\left(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]}\right) \end{cases}$$



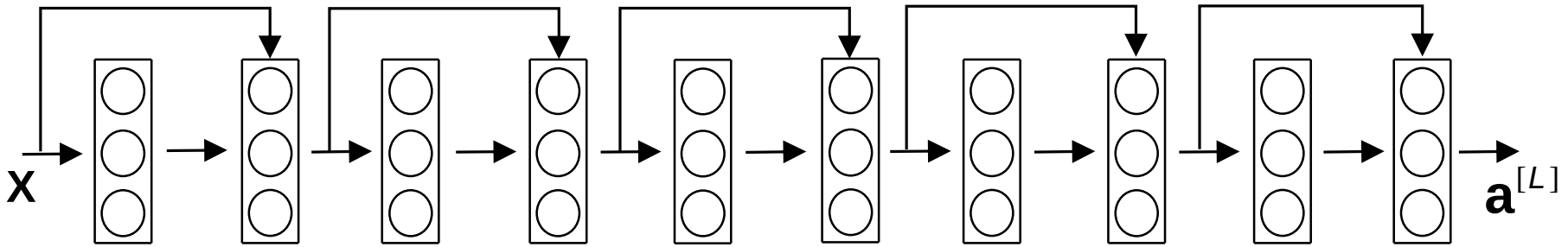
  - Observe que $\mathbf{z}^{[l+2]}$ tem que ter a mesma dimensão de $\mathbf{a}^{[l]}$.

RNAs de referência - ResNet

- Uma RNA residual é criada com blocos residuais.
- RNA convencional:

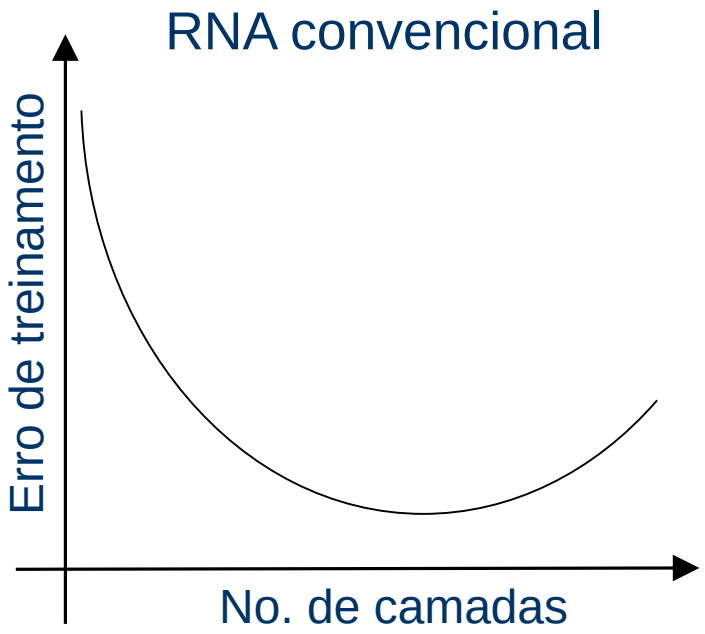


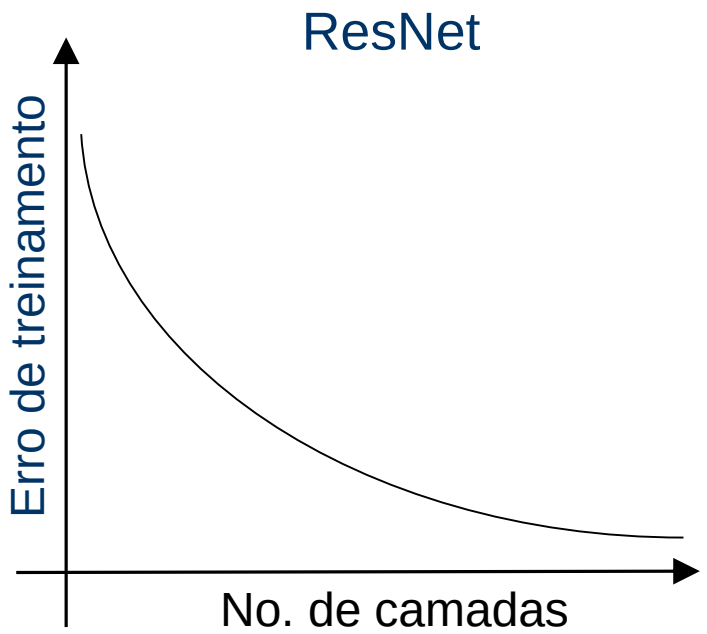
- Adiciona-se os atalhos na RNA convencional e tem-se a RNA residual:



RNAs de referência - ResNet

- Quanto mais profunda é uma RNA mais difícil de ser treinada.
- O erro de treinamento de uma RNA varia em função do número de camadas da rede.





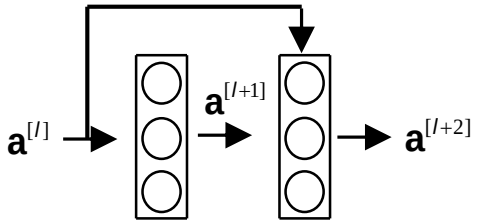
RNAs de referência - ResNet

- Na RNA convencional o erro diminui e depois aumenta com o aumento do número de camadas.
- Na RNA residual o erro somente diminui com o aumento do número de camadas.
- Uma RNA residual pode ter 1.000 ou mais camadas que não tem problema de treinamento.
- Porém existe um limite onde o aumento do número de camadas não diminui mais o erro, ou seja, o erro se estabiliza em função do número de camadas quando esse número é muito grande.

RNAs de referência - ResNet

- Porque o treinamento de uma RNA residual funciona muito bem?
  - As equações da saída do bloco podem ser escritas como sendo:

$$\mathbf{a}^{[l+2]} = g^{[l+2]}\left(\mathbf{z}^{[l+2]} + \mathbf{a}^{[l]}\right)$$
$$\mathbf{a}^{[l+2]} = g^{[l+2]}\left(\mathbf{W}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} + \mathbf{a}^{[l]}\right)$$



  - Considerando que todas as camadas do bloco residual tem função de ativação ReLu, então, se $\mathbf{W}^{[l+2]} = \mathbf{b}^{[l+2]} = 0$, temos que:

$$\mathbf{a}^{[l+2]} = \mathbf{a}^{[l]}$$

  - Ou seja, se nada foi aprendido a mais na camada *l*+2, então a saída calcula pelo bloco residual é uma função identidade.

RNAs de referência - ResNet

- Imagine que adicionamos um bloco residual na última camada de uma RNA.
- Como a função identidade é fácil de aprender pelo bloco residual, temos a seguinte situação:
  - Adicionar um bloco residual na última camada de uma RNA não altera muito essa RNA, porque a função identidade é facilmente aprendida pelo bloco residual;
  - O pior que pode acontecer é o bloco residual não fazer nada e calcular a função identidade;
  - Blocos residuais sempre aprendem alguma informação que melhora um pouco o desempenho da RNA $\Rightarrow$ adicionar vários blocos residuais pode resultar em uma grande melhoria da RNA.

➢ É muito difícil treinar uma RNA muito profunda sem blocos residuais.

RNAs de referência - ResNet

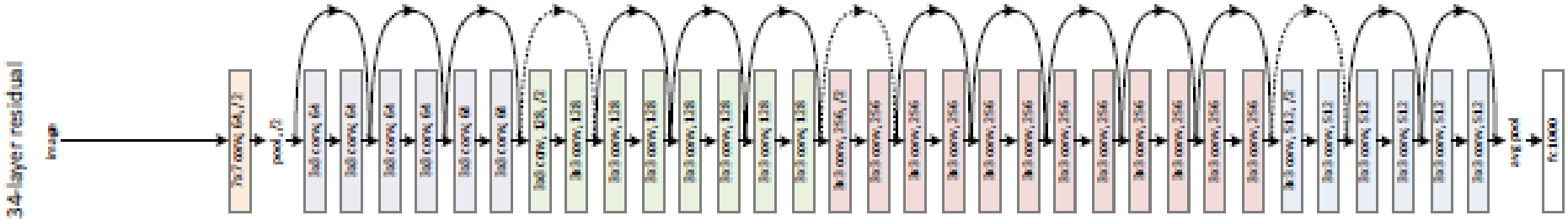
- Existe uma restrição grande nos blocos residuais:
  - $\mathbf{a}^{[l+2]}$ tem que ter a mesma dimensão de $\mathbf{a}^{[l]}$.
  - Isso implica em princípio no uso de operações de convolução que preservam as dimensões (“same convolution”).
- De fato essa restrição pode ser eliminada alterando as equações do bloco de forma a adicionar uma matriz extra de pesos para ajustar a dimensão:

$$\mathbf{a}^{[l+2]} = g^{[l+2]}\left(\mathbf{W}^{[l+2]}\mathbf{a}^{[l+1]} + \mathbf{b}^{[l+2]} + \mathbf{W}_s\mathbf{a}^{[l]}\right)$$

  $\mathbf{W}_s$ = matriz de pesos para ajuste de dimensões $\Rightarrow$ adiciona mais parâmetros que devem ser treinados.
  - Por exemplo se dimensão de $\mathbf{a}^{[l]}$ é 256 e de $\mathbf{a}^{[l+2]}$ é 128, então a dimensão de $\mathbf{W}_s$ é 128x256.

RNAs de referência - ResNet

- Arquitetura da ResNet de He et al., Deep residual networks for image recognition, 2015:



Convolução usando filtro 1x1

- Referência ⇒ Lin et al., Network in network, 2013.
- Convolução 1x1 representa realizar a operação de convolução de uma imagem por um filtro de dimensão 1x1.
- Se tivermos um tensor de um único canal:

1	2	3	6	5	8
3	5	5	1	3	4
2	1	3	4	9	3
4	7	8	5	7	9
1	5	3	7	4	8
5	4	9	8	3	5

6x6x1

* [2] =

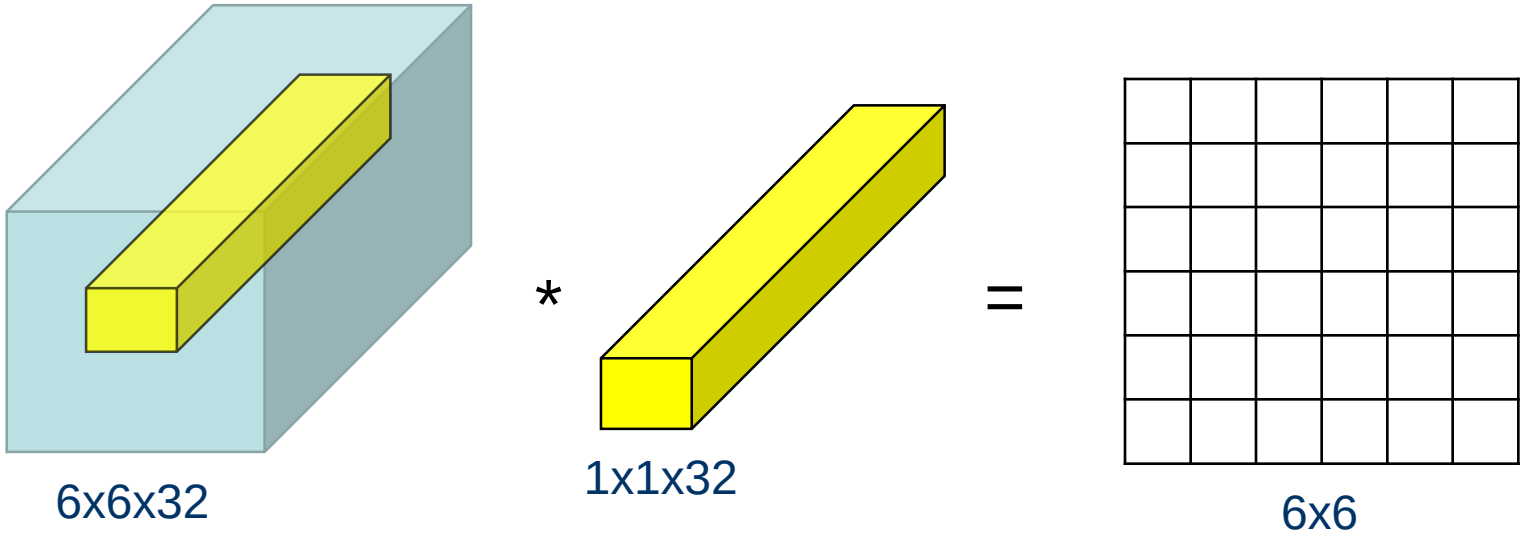
2	4	6	12	10	16
6	10	10	2	6	8
4	2	6	8	18	6
8	14	16	10	14	18
2	10	6	14	8	16
10	8	18	16	6	10

6x6x1

➢ Essa operação consiste simplesmente de uma multiplicação por uma constante.

Convolução usando filtro 1x1

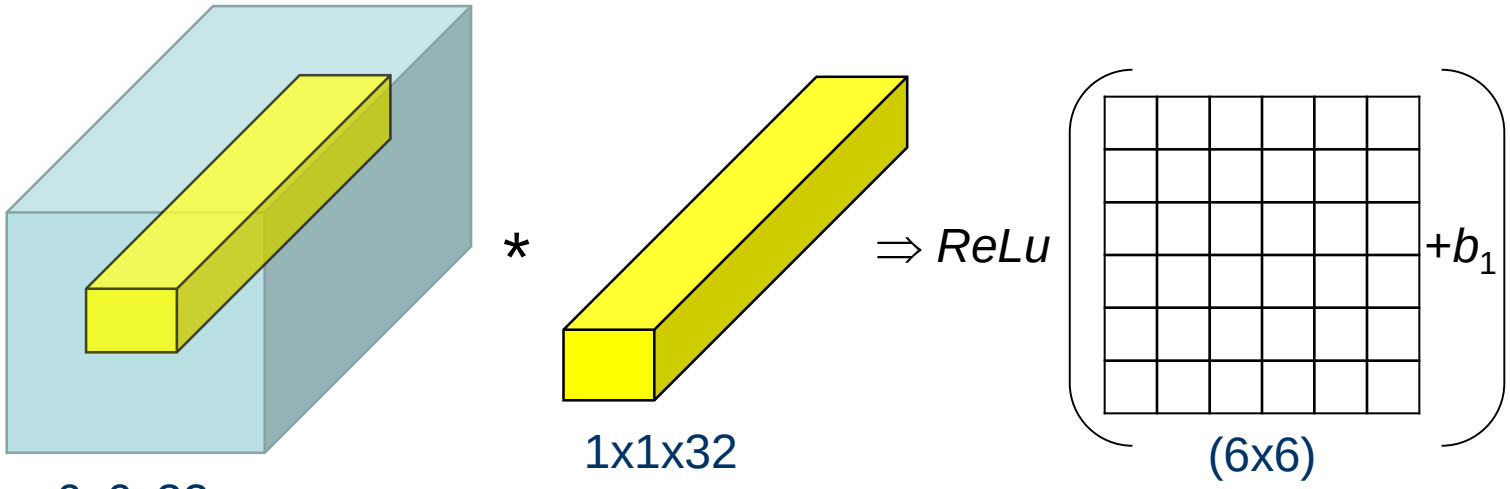
- Se tivermos um tensor de mais de um canal:



➢ Essa operação consiste do cálculo da média ponderada dos pixels de todos os canais, onde os pesos são os parâmetros do filtro.

Camada convolucional com filtro 1x1

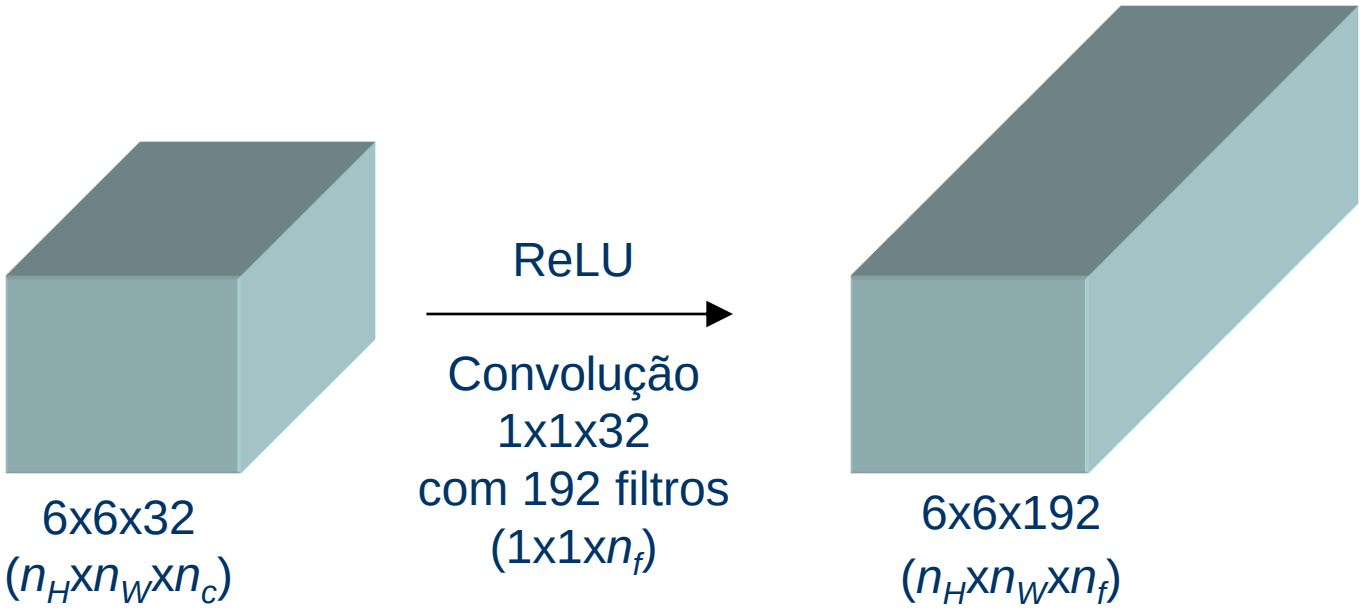
- Camada convolucional com filtros de dimensão 1x1 $\Rightarrow$ após a multiplicação dos pesos dos filtros pelos valores do tensor em todos os canais, soma uma constante e aplica uma função de ativação.



- Pode imaginar que essa operação consiste de um neurônio que recebe 32 entradas, multiplica por pesos, soma um viés e aplica uma função de ativação.

Camada convolucional com filtro 1x1

- Se existirem vários filtros 1x1, então, o resultado da camada será um tensor com largura e altura iguais ao tensor original e número de canais igual ao número de filtros.



➢ Convolução 1x1 ⇒ preserva largura e altura.

Camada convolucional com filtro 1x1

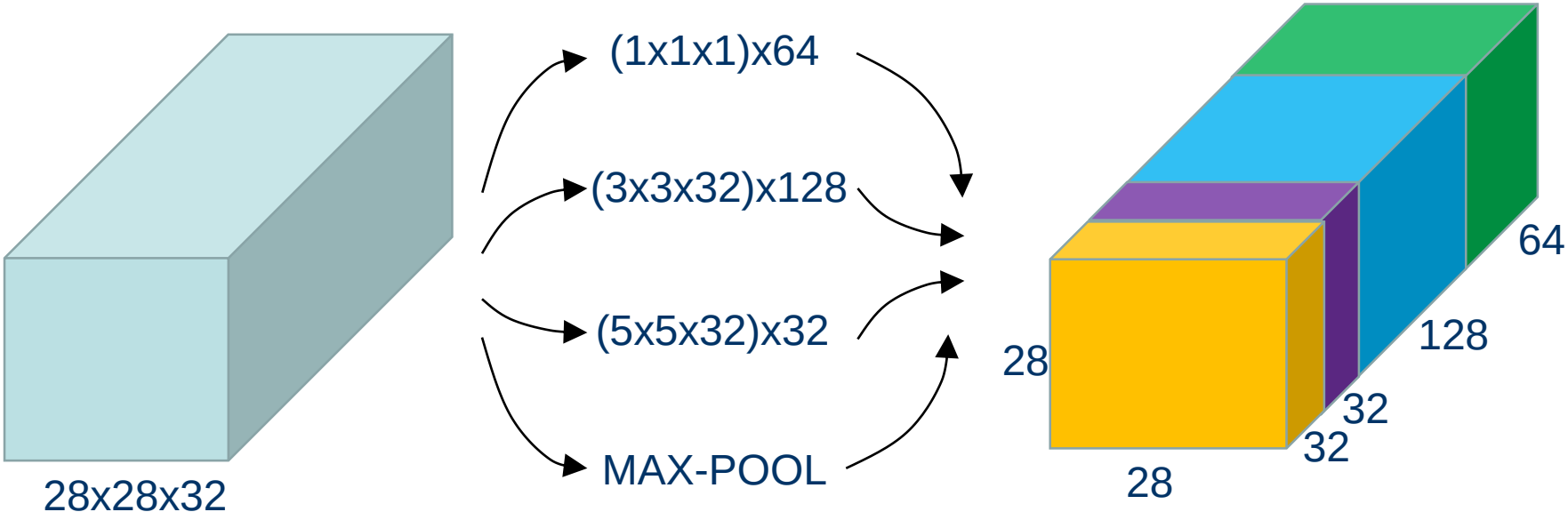
- Camada convolucional com filtro 1x1 aplica um cálculo não trivial $\Rightarrow$ age como se fosse uma camada densa.
- Como tem-se uma “camada densa” aplicada em cada elemento da largura e altura do tensor $\Rightarrow$ essa operação também é chamada rede dentro de rede.
- Utilidade da convolução com filtro 1x1:
  - Forma eficiente para diminuir número de canais quando se torna muito grande.
  - Muito eficiente para detectar novas características.
- Convolução 1x1 é usada nas RNAs Inceptions.

RNAs de referência – RNA Inception

- Referência ⇒ Szegedy et al., Going deeper with convolutions, 2014.
- Uma RNA Inception possui na mesma camada filtros com dimensões diferentes e tipo de operações diferentes.
- No lugar de escolher uma única operação por camada pode-se incluir várias operações diferentes.

RNAs de referência – RNA Inception

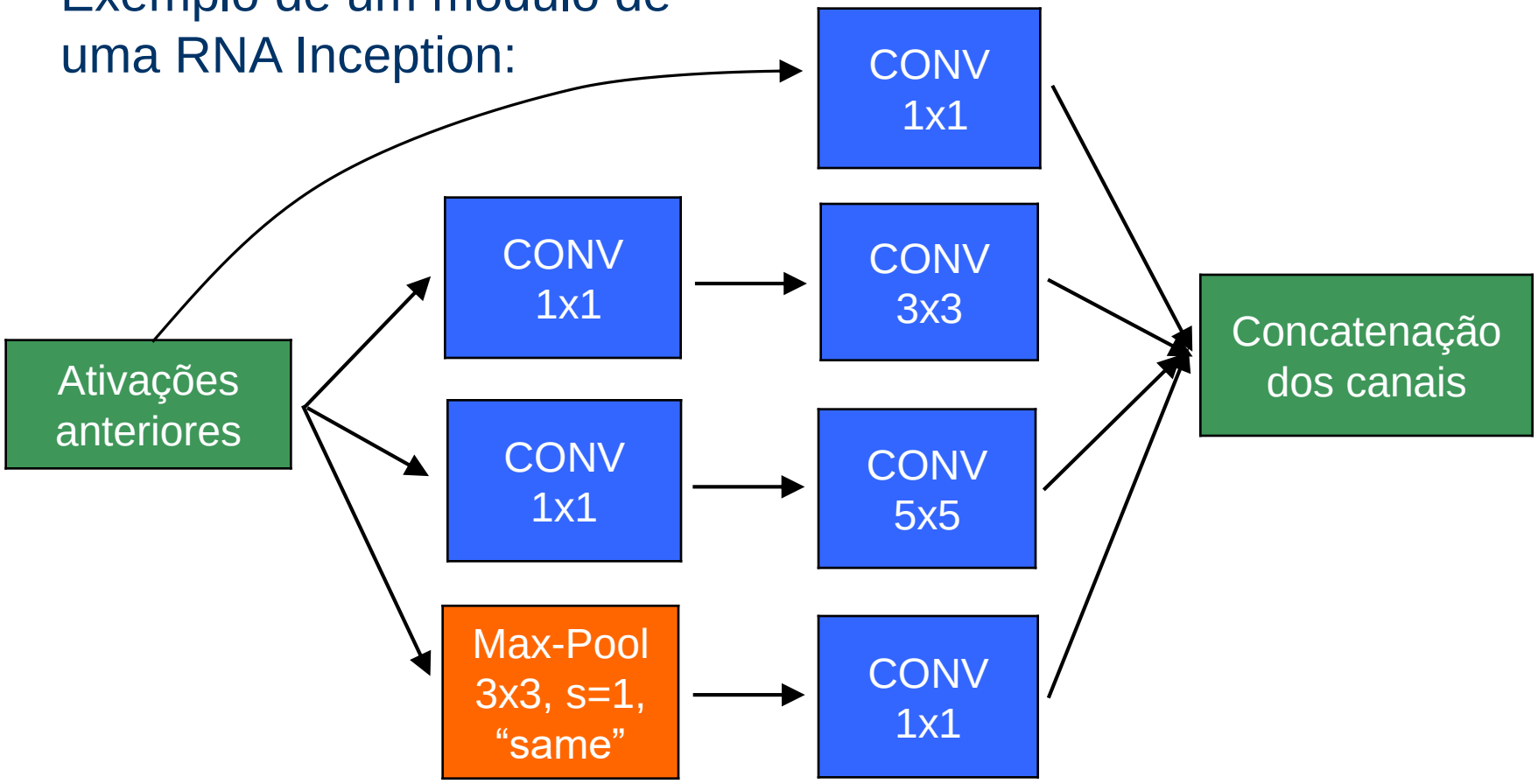
- Exemplo de uma camada com várias operações:



  - Largura e altura das saídas de todas operações precisam ser iguais;
  - Todas operações de convolução são do tipo “same” com stride igual a 1;
  - Resultados são empilhados na saída;
  - No. de canais na saída é a soma do no. de canais de cada operação (256).

RNAs de referência – RNA Inception

- Exemplo de um módulo de uma RNA Inception:

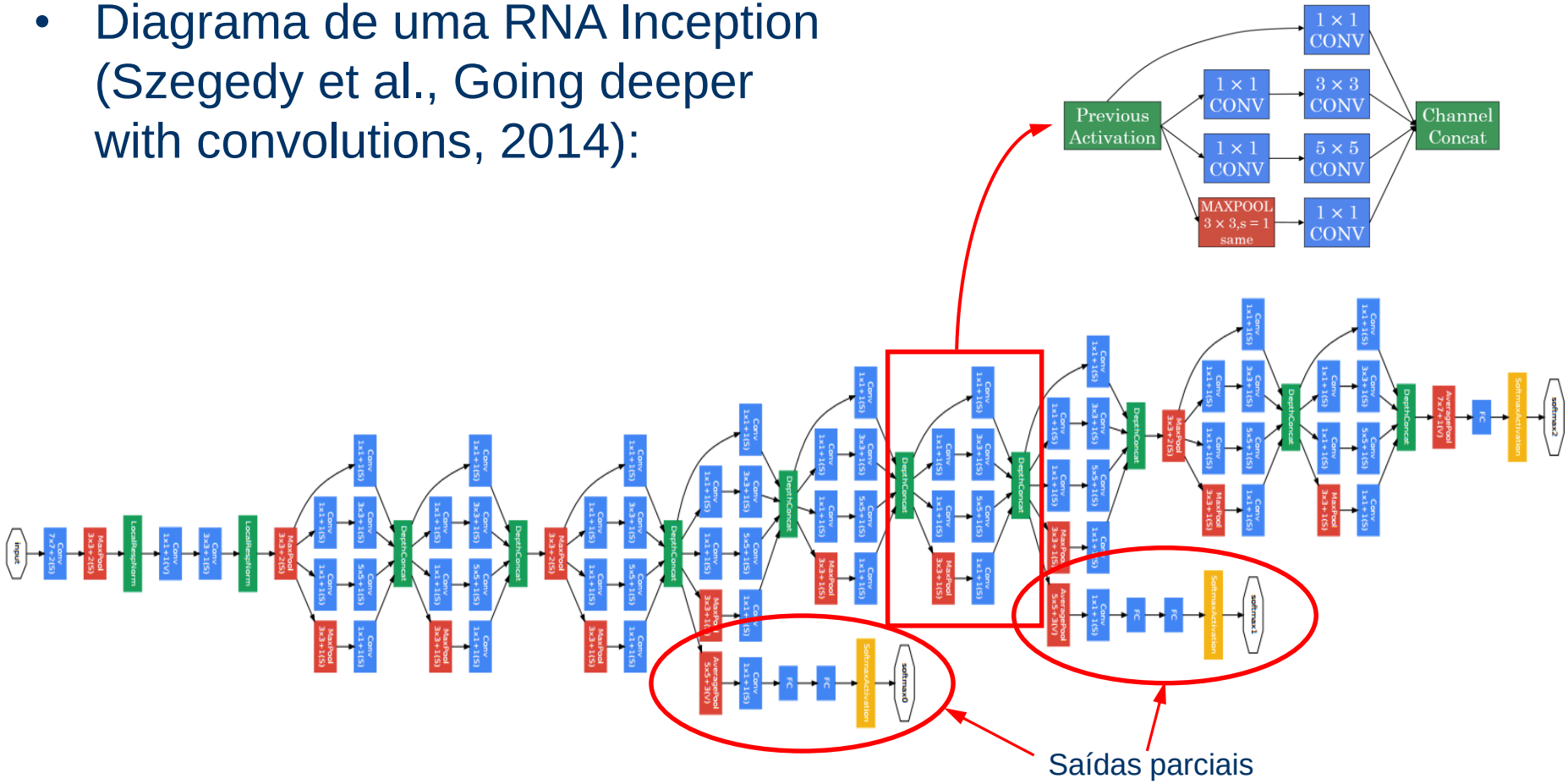


RNAs de referência – RNA Inception

- Existem diferentes tipos de módulos inception $\Rightarrow$ mas o conceito geral é o mesmo.
- Módulo de uma RNA Inception:
  - Vários tipos de filtros e operações no módulo;
  - Os resultados de todos os filtros tem a mesma largura e mesma altura;
  - Operação max-pooling é usada com stride $s = 1$, e padding $p \neq 0 \Rightarrow$ para preservar largura e altura da saída.
- RNA Inception:
  - Composta por vários módulos inceptions;
  - Repete os módulos inceptions várias vezes com pequenas modificações;
  - Possui saídas parciais $\Rightarrow$ cuja função é auxiliar na prevenção de problemas de overfitting (uma forma de regularização).

RNAs de referência – RNA Inception

- Diagrama de uma RNA Inception (Szegedy et al., Going deeper with convolutions, 2014):

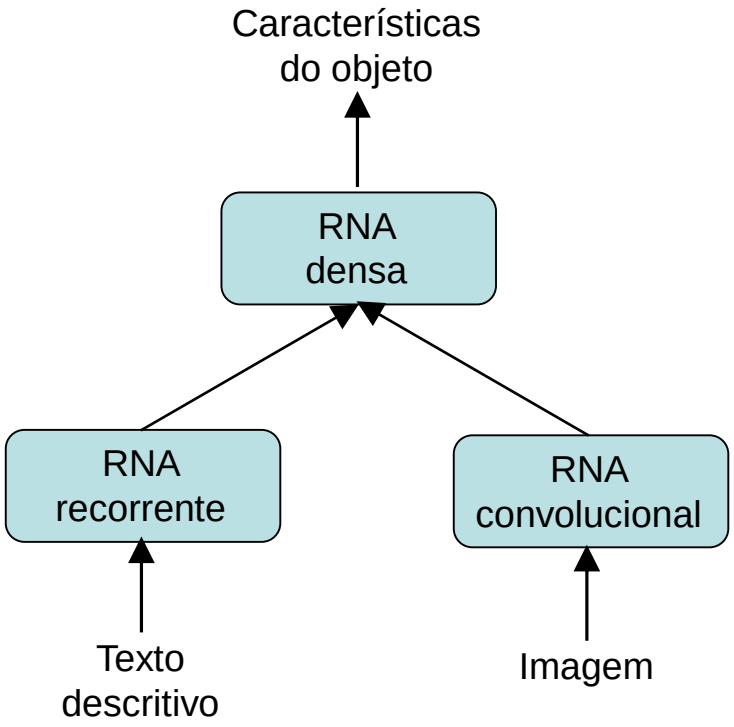


RNAs complexas

- Muitas aplicações exigem RNAs mais complexas do que as que já estudamos, como por exemplo:
  - Detecção e localização de objetos em imagens $\Rightarrow$ 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) $\Rightarrow$ nesse caso a RNA tem dois ramos iguais (RNA Siamesa);
  - Transferência de estilo $\Rightarrow$ nesse caso tem-se duas imagens que são processadas em paralelo por duas RNAs diferentes e as saídas são comparadas para gerar o erro de treinamento.

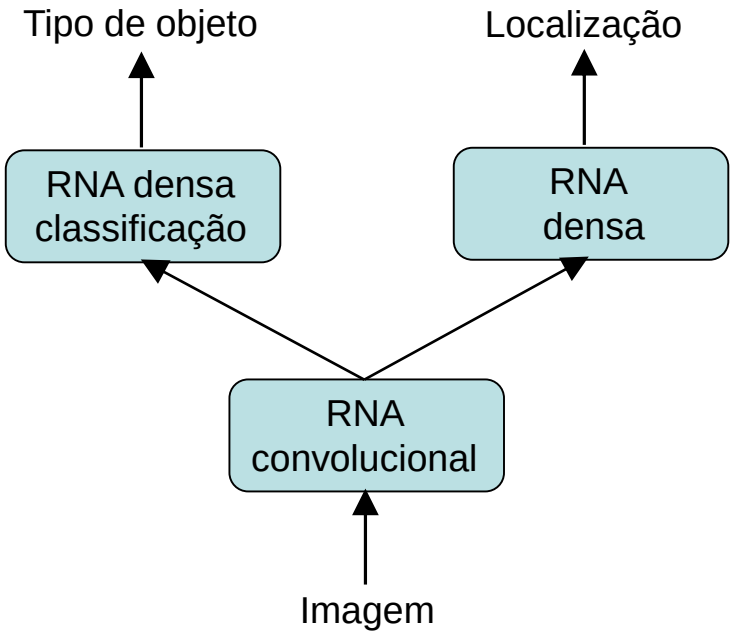
RNAs complexas

- Exemplo de RNA com diferentes tipos de entradas:



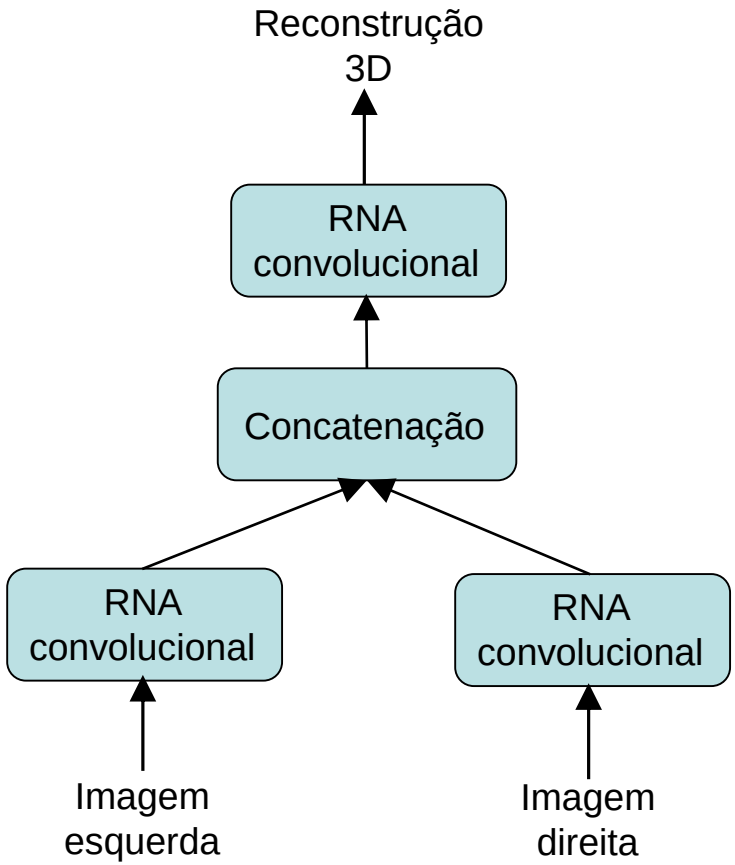
RNAs complexas

- Exemplo de RNA com diversos tipos de saídas:



RNAs complexas

- Exemplo de RNA com mais de um ramo (siamesa):



Classe Funcional do Keras

- RNAs Residual, Inception e as “complexas” possuem fluxo de informação não sequencial.
- Modelo sequencial do Keras não serve para essas RNAs.
- Para configurar uma RNA com fluxo de informação não sequencial $\Rightarrow$ 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.

Classe Funcional do Keras

- 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,), activation='relu')
A = Dense(32, activation='relu')(X)
Y = Dense(32, activation='softmax')(A)

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

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

Classe Funcional do Keras

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

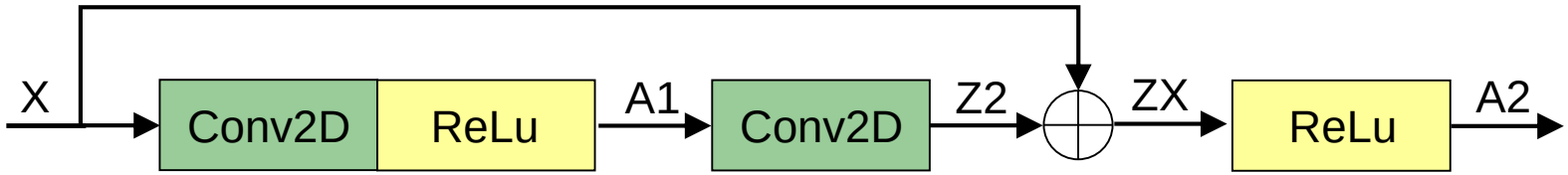
```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras import layers
from tensorflow.keras import Input

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

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

Bloco residual com Keras

- Configuração de um bloco residual com camadas convolucionais dentro de uma função (evita repetição de comandos):




```
def bloco_residual(X,n):
    # Entradas: X -> tensor de entrada
    #           n -> número de filtros
    # Saída: Y -> tensor de saída

    A1 = layers.Conv2D(n, (3,3), strides=1,
                       padding='same', activation='relu')(X)
    Z2 = layers.Conv2D(n, (3,3), strides=1,
                       padding='same', activation='linear')(A1)
    # Adiciona os dois ramos
    ZX = layers.Add()([Z2, X])
    A2 = layers.Activation('relu')(ZX)
    # Retorna saída
    return A2
```

RNA residual com keras

- Exemplo de configuração de uma RNA com 2 blocos residuais seguidos de uma camada softmax com 6 classes:

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

# Definição da entrada
Xshape = X_train.shape[1:4] # X_train tensor de imagens de treinamento
X = Input(shape=Xshape)

# Camada convolucional para ajustar número de canais
X0 = layers.Conv2D(16, (3,3), strides=1, padding='same',
                   activation='relu')(X)

# Primeiro bloco residual com 16 filtros
X1 = bloco_residual(X0,16)

# Camada convolucional para ajustar número de canais
X2 = layers.Conv2D(32, (3,3), strides=1, padding='same',
                   activation='relu')(X1)

(continua)
```

RNA residual com Keras

- Continuação:

```
# Camada de max-pooling
X2 = layers.MaxPooling2D((2, 2))(X2)

# Segundo bloco residual com 32 filtros
X2 = bloco_residual(X2,32)

# Camada de max-pooling
X2 = layers.MaxPooling2D((2, 2))(X2)

# Camada de Flattening
X3 = layers.Flatten()(X2)

# Camada densa de classificação com 6 classes
Y = layers.Dense(6, activation='softmax')(X3)

# Criação da RNA
rna = Model(X, Y)

# Mostra resumo da RNA
rna.summary()

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

RNA residual com Keras

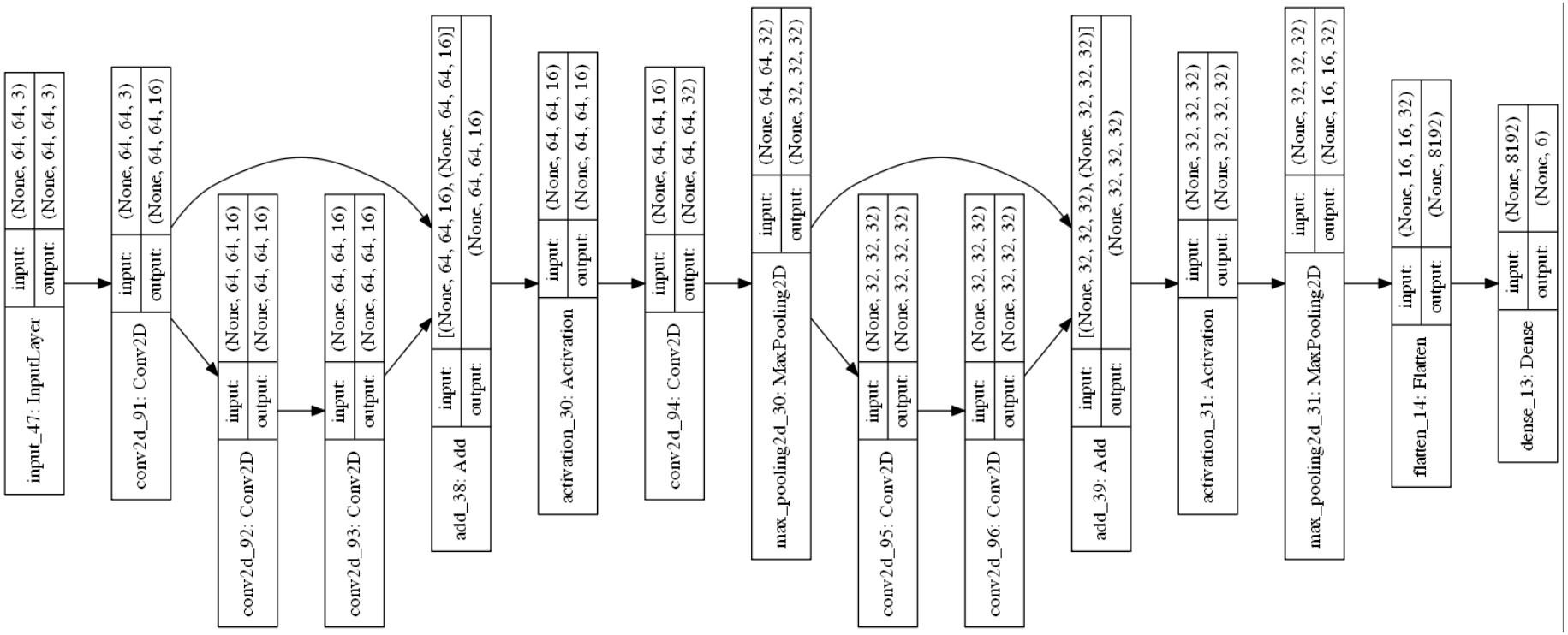
- Resumo da RNA residual:

Layer	(type)	Output Shape	Param#	Connected to
input_1	(InputLayer)	(None, 64, 64, 3)	0	
conv2d_1	(Conv2D)	(None, 64, 64, 16)	448	input_1[0][0]
conv2d_2	(Conv2D)	(None, 64, 64, 16)	2320	conv2d_1[0][0]
conv2d_3	(Conv2D)	(None, 64, 64, 16)	2320	conv2d_2[0][0]
add_1	(Add)	(None, 64, 64, 16)	0	conv2d_3[0][0] conv2d_1[0][0]
activation_1	(Activation)	(None, 64, 64, 16)	0	add_1[0][0]
conv2d_4	(Conv2D)	(None, 64, 64, 32)	4640	activation_1[0][0]
max_pooling2d_1	(MaxPooling2D)	(None, 32, 32, 32)	0	conv2d_4[0][0]
conv2d_5	(Conv2D)	(None, 32, 32, 32)	9248	max_pooling2d_1[0][0]
conv2d_6	(Conv2D)	(None, 32, 32, 32)	9248	conv2d_5[0][0]
add_2	(Add)	(None, 32, 32, 32)	0	conv2d_6[0][0] max_pooling2d_1[0][0]
activation_2	(Activation)	(None, 32, 32, 32)	0	add_2[0][0]
max_pooling2d_2	(MaxPooling2D)	(None, 16, 16, 32)	0	activation_2[0][0]
flatten_1	(Flatten)	(None, 8192)	0	max_pooling2d_2[0][0]
dense_1	(Dense)	(None, 6)	49158	flatten_1[0][0]

Total params: 77,382
Trainable params: 77,382
Non-trainable params: 0

RNA residual com Keras

- Fluxograma da RNA residual criada:



RNA residual com Keras

- Para compilar e treinar a RNA residual no Keras os comandos são os mesmos já utilizados:

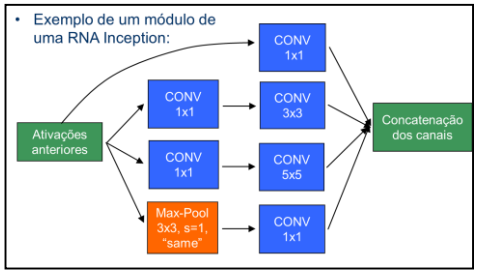
```
# Importa classe dos otimizadores do Keras
from tensorflow.keras import optimizers

# Configuração do otimizador e compilação da RNA
adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
                       decay=0.0)
rna1.compile(loss='categorical_crossentropy', metrics=['accuracy'],
             optimizer=adam)

# Treinamento da RNA
history = rna1.fit(X_train, Y_train_hot, epochs=50,
                   validation_data=(X_val, Y_val_hot), verbose=1)
```

Bloco inception com Keras

- Configuração de um módulo inception dentro de uma função:
  - Adimite que cada filtro possui *n* canais;
  - Todos os strides são iguais a 1 (padrão do keras).




```
def modulo_inception(X,n):
    # Define os ramos
    ramo_a = layers.Conv2D(n, 1, activation='relu')(x)
    ramo_b = layers.Conv2D(n, 1, activation='relu')(x)
    ramo_b = layers.Conv2D(n, 3, activation='relu',
                           padding='same')(ramo_b)
    ramo_c = layers.Conv2D(n, 1, activation='relu')(x)
    ramo_c = layers.Conv2D(n, 5, activation='relu',
                           padding='same')(ramo_c)
    ramo_d = layers.MaxPooling2D(3, strides=1, padding='same')(x)
    ramo_d = layers.Conv2D(n, 1, activation='relu',
                           padding='same')(ramo_d)
    # Une os ramos na saída do módulo
    y = layers.concatenate([ramo_a, ramo_b, ramo_c, ramo_d], axis=-1)
    # Retorna saída
    return y
```

RNA inception com Keras

- Exemplo de configuração de uma RNA com 2 blocos inceptions seguidos de uma camada softmax com 6 classes:

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

# Definição da entrada
Xshape = X_train.shape[1:4] # X_train tensor de imagens de treinamento
X = Input(shape=Xshape)

# Primeiro bloco inception com filtros de 64 canais
X1 = modulo_inception(X,64)

# Camada de pooling entre os módulos
X2 = layers.MaxPooling2D(3, strides=2)(X1)

# Segundo modulo inception com filtros de 128 canais
X3 = modulo_inception(X2,128)

(continua)
```

RNA Inception com Keras

- Continuação:

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

# Camada densa de classificação com 6 classes
Y = layers.Dense(6, activation='softmax')(X4)

# Criação da RNA
rna = Model(X, Y)

# Mostra resumo da RNA
rna.summary()

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

RNA inception com Keras

- Resumo da RNA inception:

Layer	(type)	Output Shape	Param#	Connected to
input_1	(InputLayer)	(None, 64, 64, 3)	0	
conv2d_2	(Conv2D)	(None, 64, 64, 16)	64	input_1[0][0]
conv2d_4	(Conv2D)	(None, 64, 64, 16)	64	input_1[0][0]
max_pooling2d_1	(MaxPooling2D)	(None, 64, 64, 3)	0	input_1[0][0]
conv2d_1	(Conv2D)	(None, 64, 64, 16)	64	input_1[0][0]
conv2d_3	(Conv2D)	(None, 64, 64, 16)	2320	conv2d_2[0][0]
conv2d_5	(Conv2D)	(None, 64, 64, 16)	6416	conv2d_4[0][0]
conv2d_6	(Conv2D)	(None, 64, 64, 16)	64	max_pooling2d_1[0][0]
concatenate_1	(Concatenate)	(None, 64, 64, 64)	0	conv2d_1[0][0] conv2d_3[0][0] conv2d_5[0][0] conv2d_6[0][0]
max_pooling2d_2	(MaxPooling2D)	(None, 31, 31, 64)	0	concatenate_1[0][0]
conv2d_8	(Conv2D)	(None, 31, 31, 32)	2080	max_pooling2d_2[0][0]
conv2d_10	(Conv2D)	(None, 31, 31, 32)	2080	max_pooling2d_2[0][0]
max_pooling2d_3	(MaxPooling2D)	(None, 31, 31, 64)	0	max_pooling2d_2[0][0]
conv2d_7	(Conv2D)	(None, 31, 31, 32)	2080	max_pooling2d_2[0][0]
conv2d_9	(Conv2D)	(None, 31, 31, 32)	9248	conv2d_8[0][0]

(continua)

RNA inception com Keras

- Resumo da RNA inception (continuação):

```
conv2d_11         (Conv2D)          (None, 31, 31, 32)   25632    conv2d_10[0][0]
conv2d_12         (Conv2D)          (None, 31, 31, 32)   2080     max_pooling2d_3[0][0]
concatenate_2     (Concatenate)     (None, 31, 31, 128)  0        conv2d_7[0][0]
                                                                    conv2d_9[0][0]
                                                                    conv2d_11[0][0]
                                                                    conv2d_12[0][0]
flatten_1         (Flatten)         (None, 123008)       0        concatenate_2[0][0]
dense_1           (Dense)           (None, 6)            738054   flatten_1[0][0]
==================================================================================
Total params: 790,246
Trainable params: 790,246
Non-trainable params: 0
```

RNA inception com Keras

- Fluxograma da RNA inception criada: