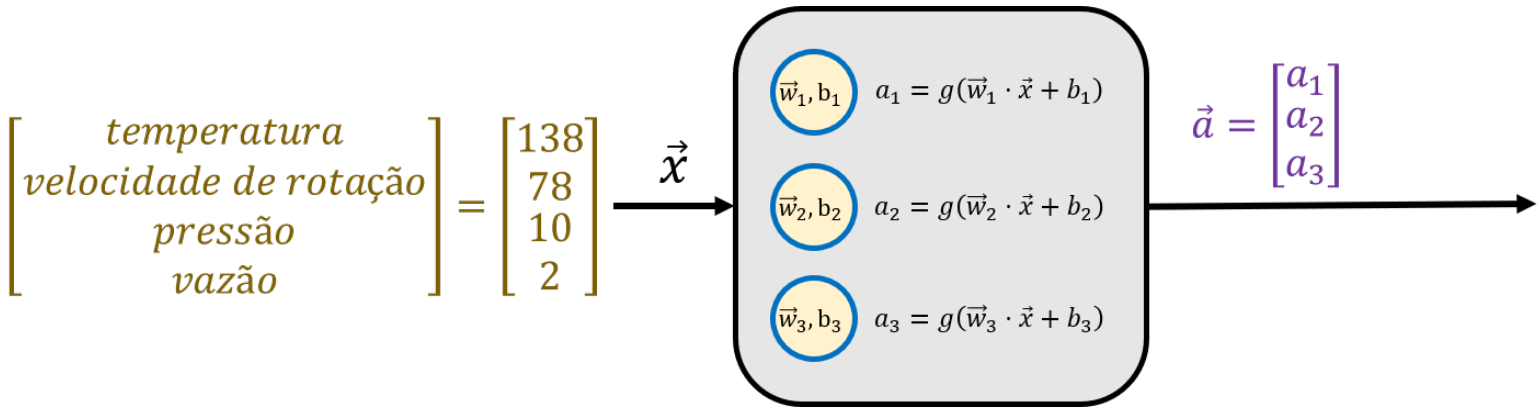
# Camadas convolucionais e *Dropout*

Esta aula constitui um tópico adicional da disciplina. Trata-se de um conteúdo opcional. Sua atividade não valerá nota e não precisa ser enviada.


## Camadas convolucionais

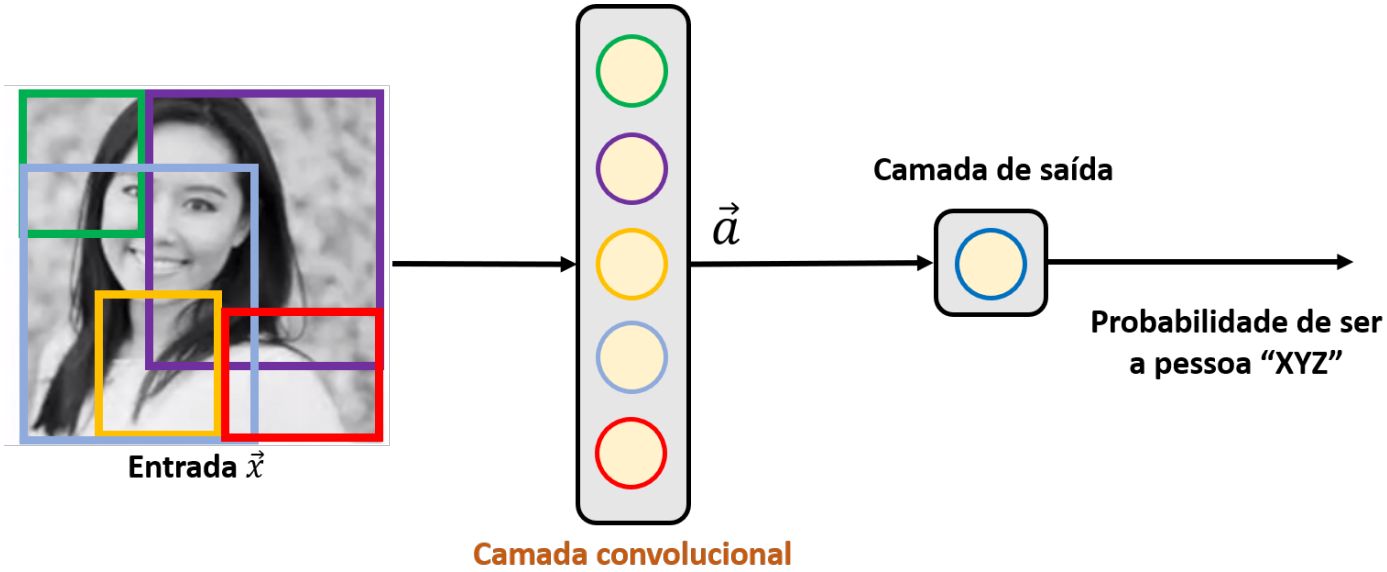
- Conforme ilustrado abaixo, em camadas do tipo **Dense**, a saída de cada neurônio é uma função de **todas as ativações** de saída vindas da camada anterior.
- No exemplo, por se tratar da Camada 1, note que as ativações de saída vindas da camada anterior são o próprio $\vec{x} = \vec{a}^{[0]}$ (características de entrada).




A seguir, veremos um exemplo de um outro tipo de camada que é possível de ser utilizada numa rede neural

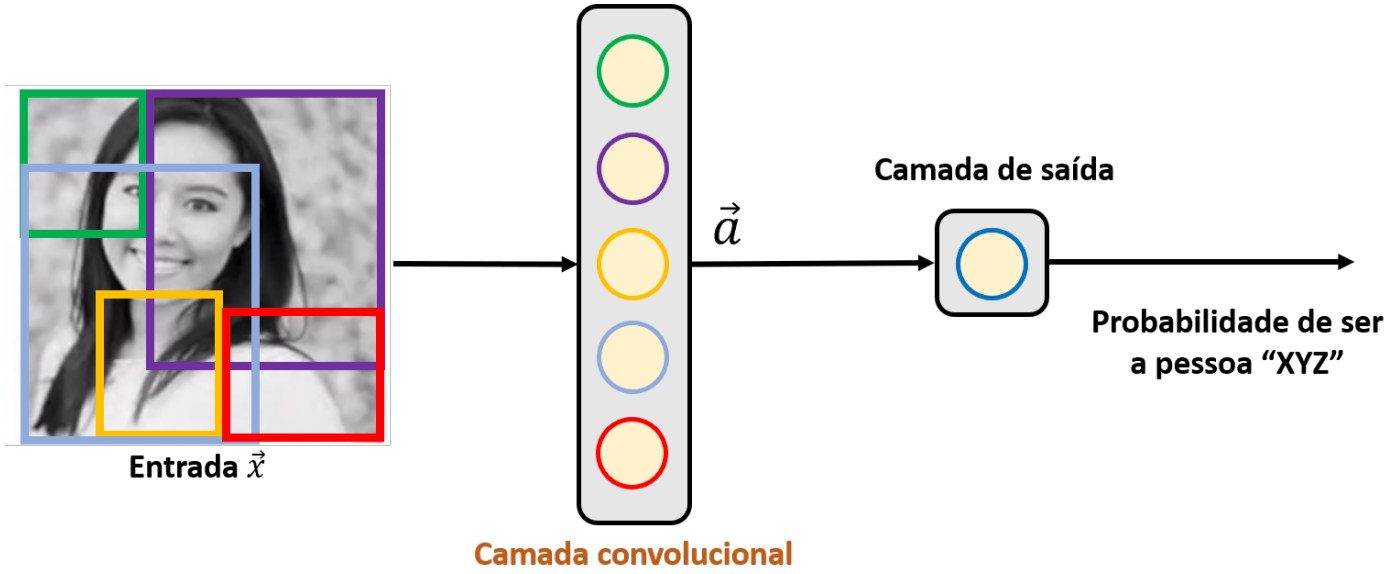
# Camadas convolucionais

- Conforme ilustrado abaixo, em camadas do tipo **Convolucionais**, a saída de cada neurônio é uma função apenas de **parte das ativações** de saída vindas da camada anterior.



# Camadas convolucionais

- Conforme ilustrado abaixo, em camadas do tipo **Convolucionais**, a saída de cada neurônio é uma função apenas de **parte das ativações** de saída vindas da camada anterior.




**Qual é a vantagem de fazermos isso?**

- O processamento torna-se mais rápido.
- Menos dados de treinamento são necessários, já que menos parâmetros estarão presentes no modelo (tendência menor de ocorrer *overfitting*)

**Observação**

- Se a sua rede neural possui muitas camadas convolucionais, então ela pode também ser chamada de **Rede neural convolucional**

## Desvantagens de usar redes neurais clássicas para classificação de imagens

- Requer uma quantidade muito grande de cálculos
- Pixels locais são tratados da mesma maneira que pixels distantes
- sensível à posição do objeto na imagem

**Pergunta:**

Como podemos detectar pequenos elementos numa imagem?

**Resposta:**

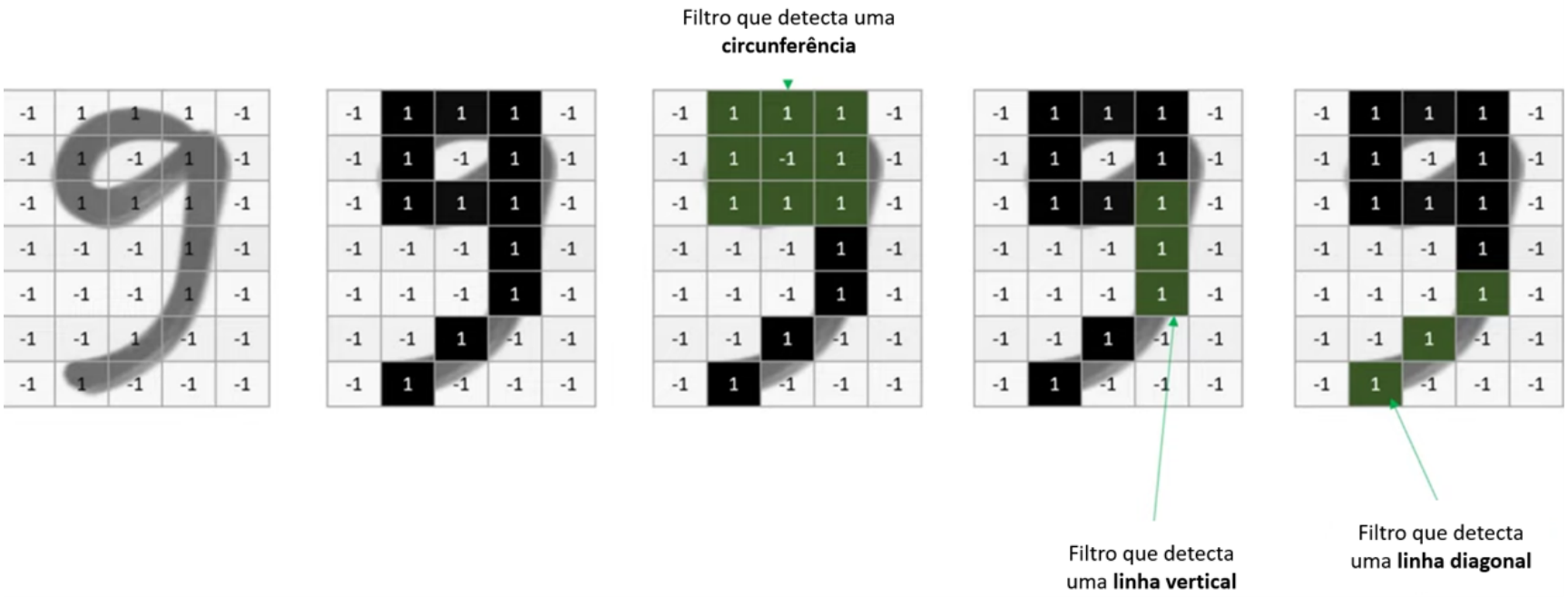
Simples! Usando filtros.

**Exemplo:**

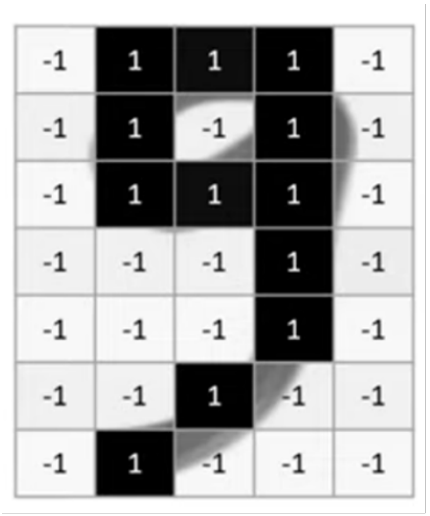
Suponha que você quer encontrar, numa imagem, onde ela possui

- uma circunferência
- uma linha vertical
- uma linha diagonal

Nesse caso, os três filtros abaixo podem ser usados.



Aplicando o filtro que detecta circunferência:




* (conv.)

Filtro que detecta uma **circunferência**

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -1 | 1 |
| 1 | 1 | 1 |

=

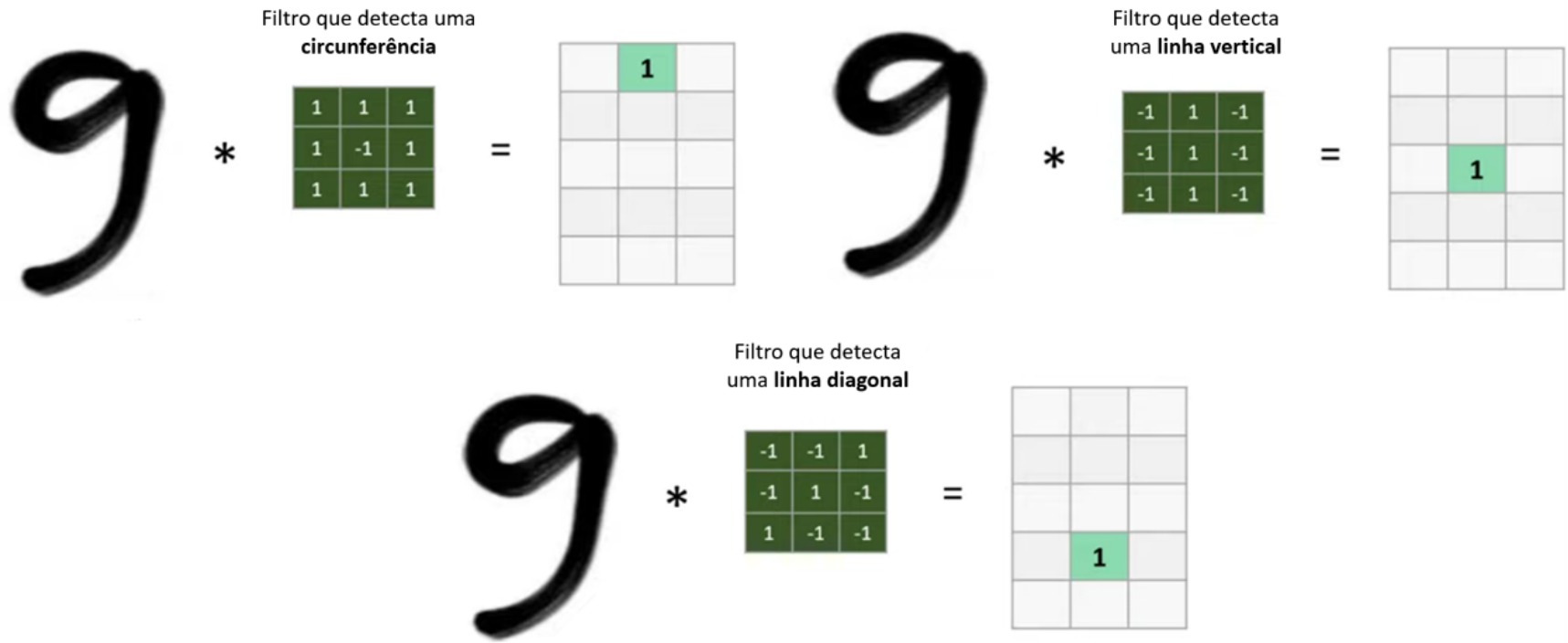
| | | |
|---|---|---|
| -0.11 | **1** | -0.11 |
| -0.55 | 0.11 | -0.33 |
| -0.33 | 0.33 | -0.33 |
| -0.22 | -0.11 | -0.22 |
| -0.33 | -0.33 | -0.33 |

**Mapa de características:** Nova imagem que destaca a posição onde está presente a circunferência

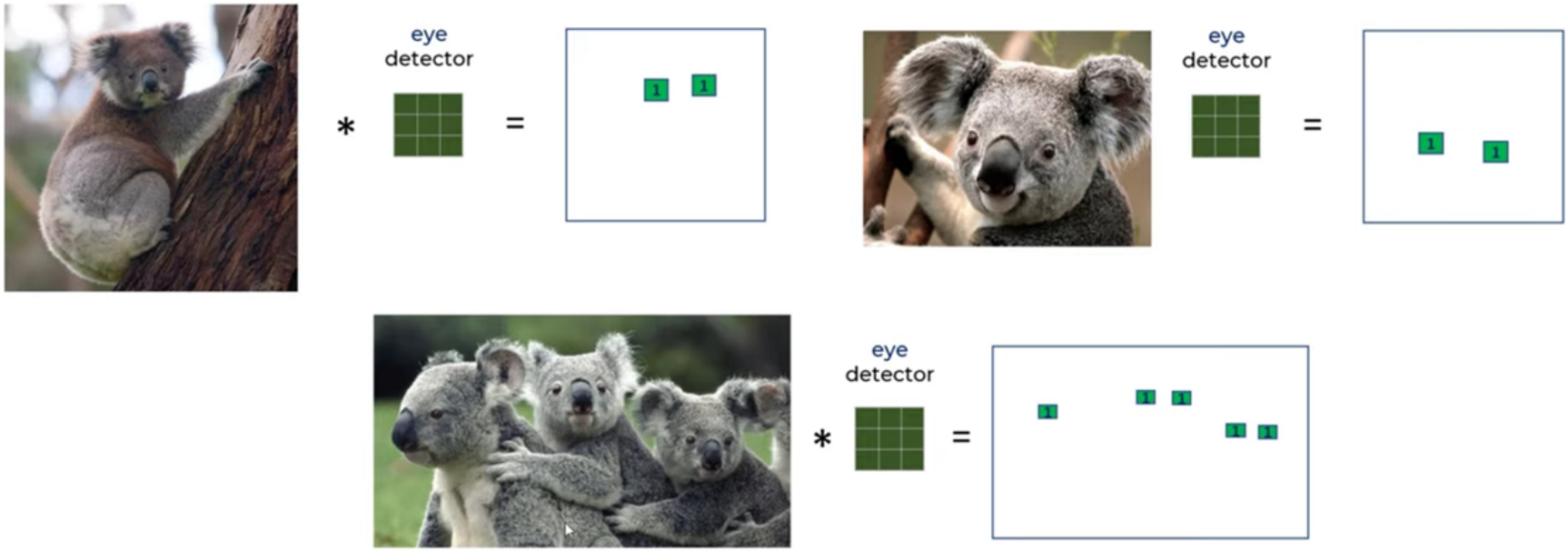
## Observações

- Note que o filtro acima foi aplicado à imagem original usando um passo de 1 unidade.
- É possível usar um passo maior. Isso reduz ainda mais a dimensionalidade da imagem resultante.

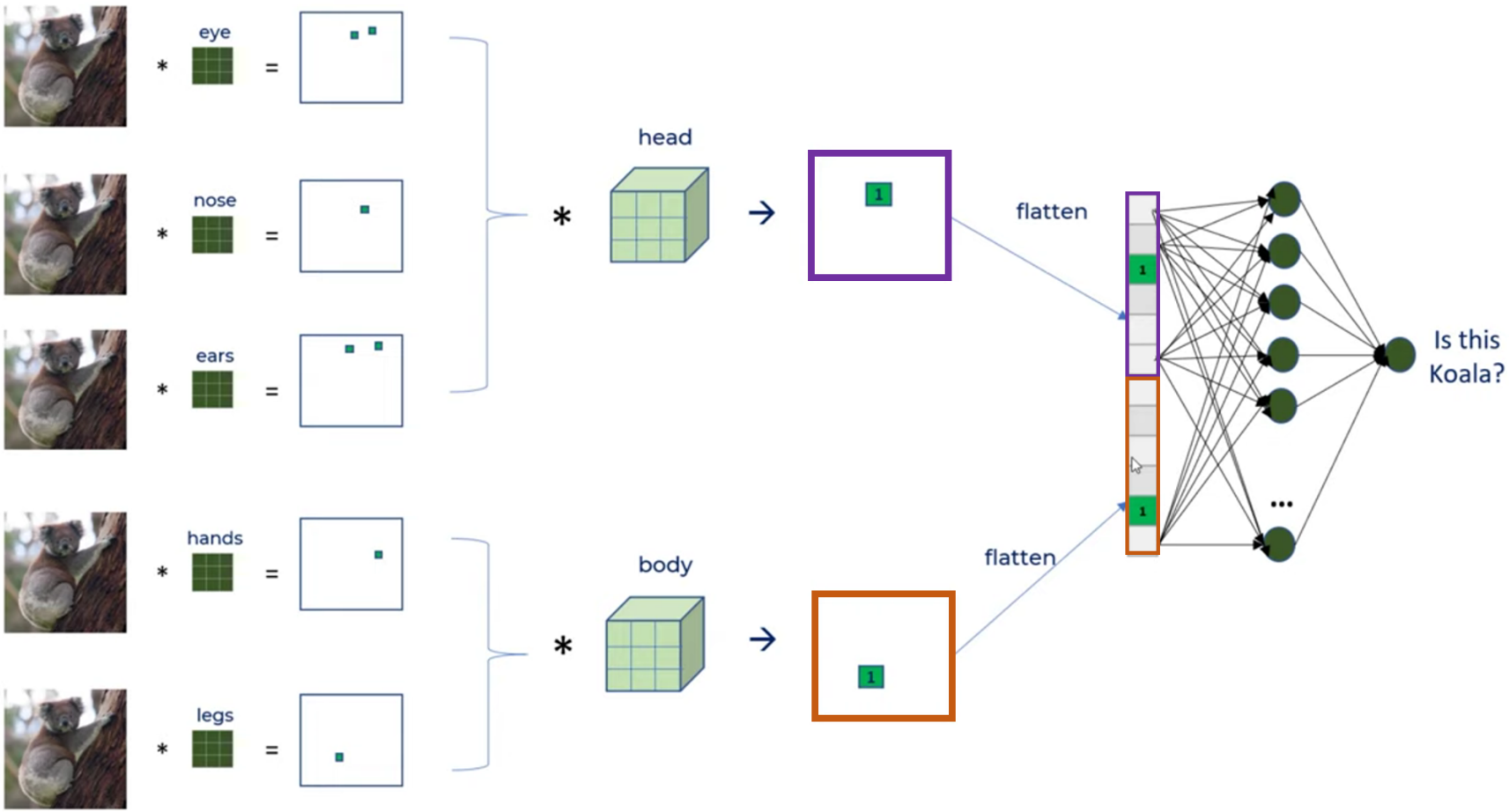
Outros filtros interessantes para identificar um 9 numa imagem:



Filtro detector de olhos:




Note que o filtro funciona independentemente da posição ou quantidade de olhos na imagem. Essa é uma das grandes vantagens das camadas convolucionais.




**Observações:**

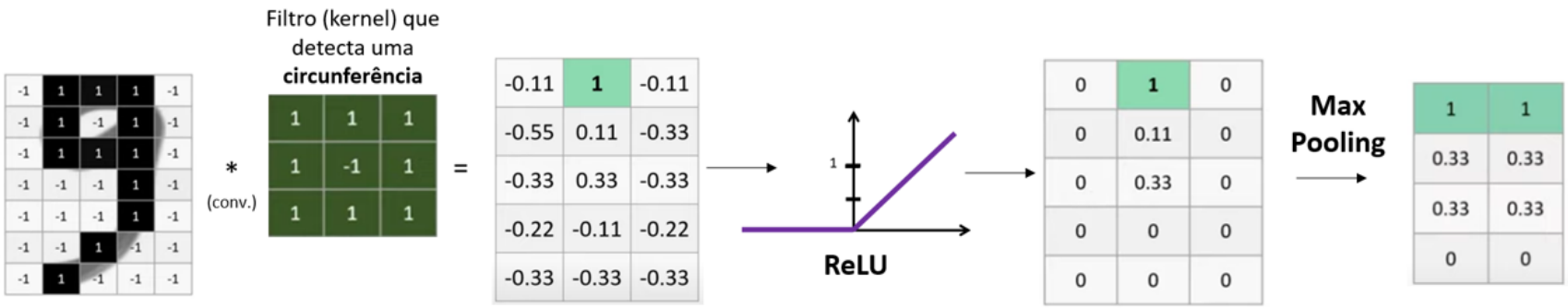
- Note que a rede acima aplica dois filtros 3D para localizar a cabeça e o corpo na imagem.
- Esses dois resultados são então "desenrolados" e colocados como entrada de uma rede neural clássica (com camadas do tipo Dense)

- Nas redes neurais clássicas, usamos funções de ativação para adicionar não linearidades matemáticas.
- Nas camadas convolucionais, também podemos fazer isso. Por exemplo, usando a função ReLU ao mapa de características obtido no slides anteriores, chegamos em:

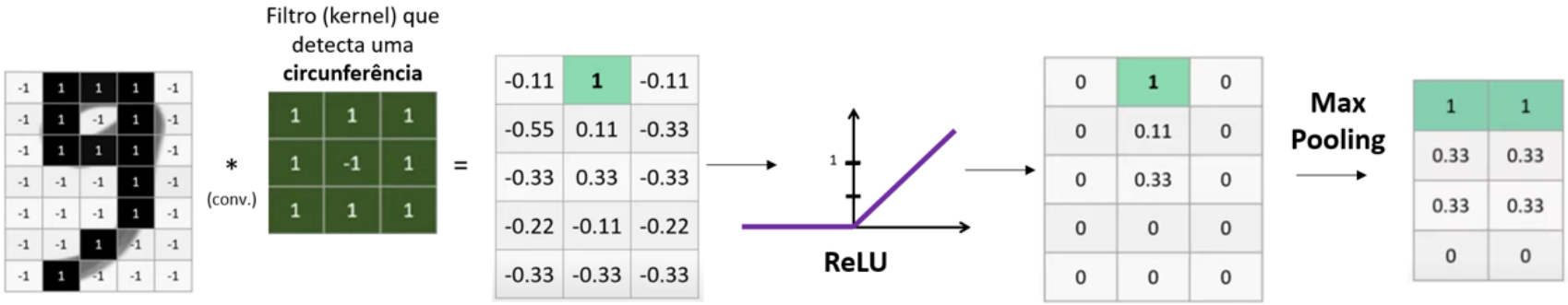

**Mais observações:**

- Caso você queira que o mapa de características tenha a mesma dimensão que a figura original, você pode fazer "preenchimento forçado" (*padding*). Para o exemplo acima, poderia, por exemplo, adicionar zeros em volta.
- Entretanto, muitas vezes queremos justamente reduzir ainda mais as dimensões do nosso problema, para que haja também redução de custo computacional. Nesse contexto, existe o chamado *pooling*.

Exemplo de filtro de *Max Pooling* 2 por 2 com passo 1:



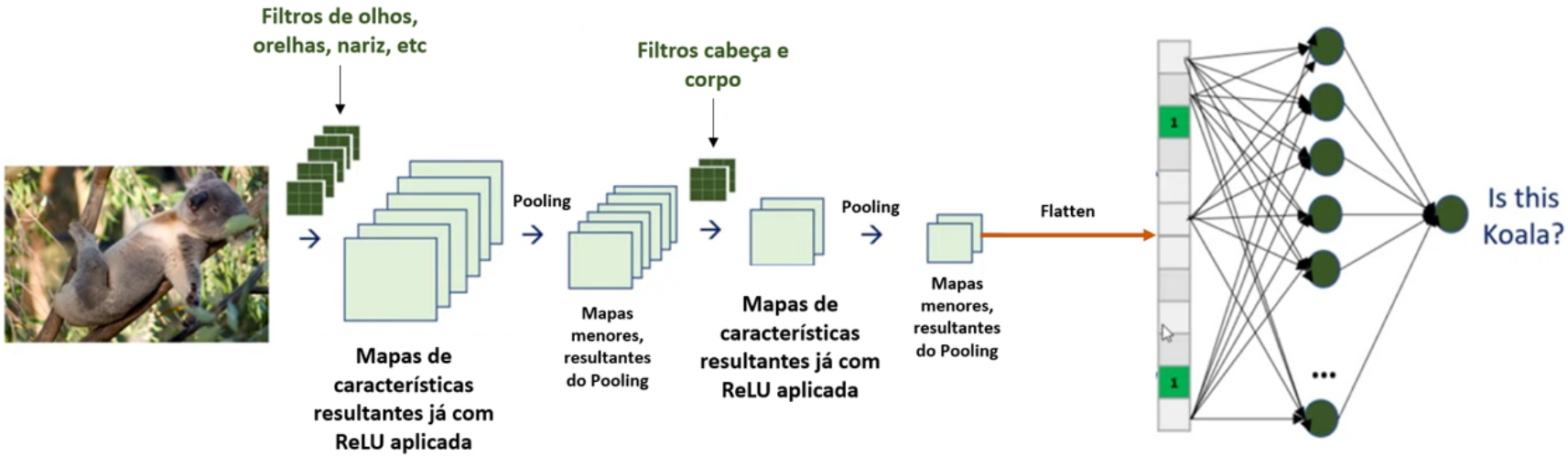
Exemplo de filtro de *Max Pooling* 2 por 2 com passo 1:



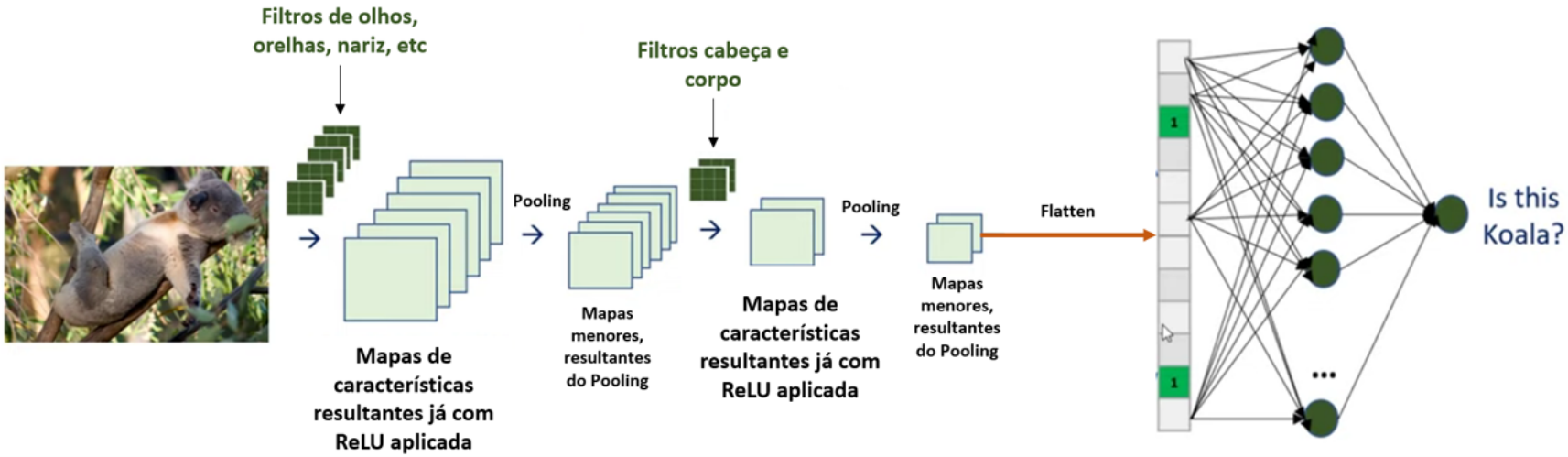

## Observações

- Ao invés de *Max Pooling*, poderíamos também usar *average Pooling*.
- Redução de dimensionalidade vem acompanhada de redução de parâmetros a serem estimados.

Rede convolucional completa:



Rede convolucional completa:




Observações:

- Os valores dos filtros acima são parâmetros que devem ser treinados.
- São hiperparâmetros: O tamanho dos filtros, o passo usado para movê-los quando do cálculo da convolução, o tipo de Pooling (Max ou average), e as funções de ativação.
- Para mais detalhes sobre redes convolucionais, assistir `https://www.youtube.com/watch?v=zfiSAzpy9NM&t=1289s`

## Dropout

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

## Passo-a-passo para implementar o Dropout:

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

## Passo-a-passo para implementar o Dropout:

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

**Passo-a-passo para implementar o Dropout:**

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).
2. Isso força a rede a não depender demais de neurônios específicos, incentivando os neurônios restantes a trabalhar de forma mais independente e colaborativa.

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

**Passo-a-passo para implementar o Dropout:**

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).
2. Isso força a rede a não depender demais de neurônios específicos, incentivando os neurônios restantes a trabalhar de forma mais independente e colaborativa.
3. No final, todos os neurônios são usados durante a inferência (quando o modelo faz previsões), mas suas conexões são ajustadas para refletir a média de quando estavam "ligados" ou "desligados" no treinamento.

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

## Passo-a-passo para implementar o Dropout:

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).
2. Isso força a rede a não depender demais de neurônios específicos, incentivando os neurônios restantes a trabalhar de forma mais independente e colaborativa.
3. No final, todos os neurônios são usados durante a inferência (quando o modelo faz previsões), mas suas conexões são ajustadas para refletir a média de quando estavam "ligados" ou "desligados" no treinamento.

## Observações finais

- A rede aprende representações mais gerais e confiáveis.

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

## Passo-a-passo para implementar o Dropout:

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).
2. Isso força a rede a não depender demais de neurônios específicos, incentivando os neurônios restantes a trabalhar de forma mais independente e colaborativa.
3. No final, todos os neurônios são usados durante a inferência (quando o modelo faz previsões), mas suas conexões são ajustadas para refletir a média de quando estavam "ligados" ou "desligados" no treinamento.

## Observações finais

- A rede aprende representações mais gerais e confiáveis.
- É controlado por um parâmetro chamado taxa de dropout, que indica a proporção de neurônios a serem desativados (exemplo: 0.2 significa que 20% dos neurônios serão desligados a cada iteração).

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

## Passo-a-passo para implementar o Dropout:

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).
2. Isso força a rede a não depender demais de neurônios específicos, incentivando os neurônios restantes a trabalhar de forma mais independente e colaborativa.
3. No final, todos os neurônios são usados durante a inferência (quando o modelo faz previsões), mas suas conexões são ajustadas para refletir a média de quando estavam "ligados" ou "desligados" no treinamento.

## Observações finais

- A rede aprende representações mais gerais e confiáveis.
- É controlado por um parâmetro chamado taxa de dropout, que indica a proporção de neurônios a serem desativados (exemplo: 0.2 significa que 20% dos neurônios serão desligados a cada iteração).

## Resumo:

- É como se você fosse o treinador de um time de futebol, e você vai tirando jogadores do seu time aleatoriamente para que eles aprendam a se adaptar quando da falta de alguns jogadores específicos.

# Dropout

O **Dropout** é uma técnica usada no treinamento de redes neurais para reduzir o risco de overfitting, que é quando o modelo aprende demais com os dados de treinamento e não generaliza bem para novos dados.

## Passo-a-passo para implementar o Dropout:

1. Em cada iteração, alguns neurônios da rede são desativados aleatoriamente (ou seja, seus valores de saída são zerados).
2. Isso força a rede a não depender demais de neurônios específicos, incentivando os neurônios restantes a trabalhar de forma mais independente e colaborativa.
3. No final, todos os neurônios são usados durante a inferência (quando o modelo faz previsões), mas suas conexões são ajustadas para refletir a média de quando estavam "ligados" ou "desligados" no treinamento.

## Observações finais

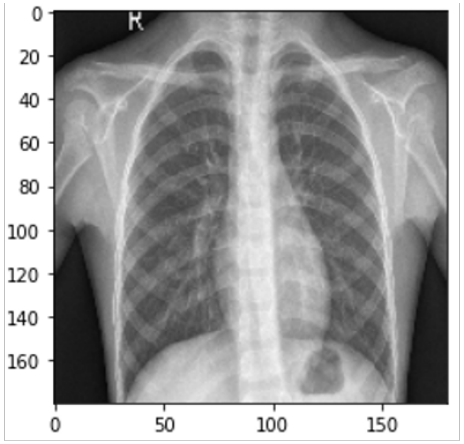
- A rede aprende representações mais gerais e confiáveis.
- É controlado por um parâmetro chamado taxa de dropout, que indica a proporção de neurônios a serem desativados (exemplo: 0.2 significa que 20% dos neurônios serão desligados a cada iteração).

## Resumo:

- É como se você fosse o treinador de um time de futebol, e você vai tirando jogadores do seu time aleatoriamente para que eles aprendam a se adaptar quando da falta de alguns jogadores específicos.
- Assim, fazemos com que "a equipe trabalhe com membros diferentes em cada treino", garantindo que o aprendizado não dependa de poucos indivíduos específicos, mas sim do esforço coletivo.

## Um exemplo complexo

# Um exemplo complexo




→ Probabilidade de ser pneumonia

```python
def conv_block(filters, inputs):
    x = layers.SeparableConv2D(filters, 3, activation="relu", padding="same")(inputs)
    x = layers.SeparableConv2D(filters, 3, activation="relu", padding="same")(x)
    x = layers.BatchNormalization()(x)
    outputs = layers.MaxPool2D()(x)

    return outputs


def dense_block(units, dropout_rate, inputs):
    x = layers.Dense(units, activation="relu")(inputs)
    x = layers.BatchNormalization()(x)
    outputs = layers.Dropout(dropout_rate)(x)

    return outputs
```

```python
def build_model():
    inputs = keras.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))
    x = layers.Rescaling(1.0 / 255)(inputs)
    x = layers.Conv2D(16, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(16, 3, activation="relu", padding="same")(x)
    x = layers.MaxPool2D()(x)

    x = conv_block(32, x)
    x = conv_block(64, x)

    x = conv_block(128, x)
    x = layers.Dropout(0.2)(x)

    x = conv_block(256, x)
    x = layers.Dropout(0.2)(x)

    x = layers.Flatten()(x)
    x = dense_block(512, 0.7, x)
    x = dense_block(128, 0.5, x)
    x = dense_block(64, 0.3, x)

    outputs = layers.Dense(1, activation="sigmoid")(x)

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

**De olho no código!**

# De olho no código!

Vamos agora rodar esse exemplo complexo que trata da detecção de pneumonia com base no Raio X do paciente.

Acesse o código diretamente pelo site do Keras, usando o QR code ou o link abaixo:

`https://keras.io/examples/vision/xray_classification_with_tpus/`

**Observação importante:**

Note que o site do Keras possui diversos outros exemplos interessantes de aplicações envolvendo ML. São aplicações envolvendo Aprendizado Supervisionado, Não Supervisionado, Aprendizado por Reforço, etc.

# Atividade de aula

**Parte 1**

Rode todo o código. Responda às questões nele contidas e complete-o, se necessário.

**Parte 2**

1. Explique em detalhes a arquitetura de rede neural convolucional que está sendo utilizada. Também identifique a taxa de dropout utilizada no código.