iscte
TECNOLOGIAS
E ARQUITETURA

APVC

Redes Neuronais Convolucionais

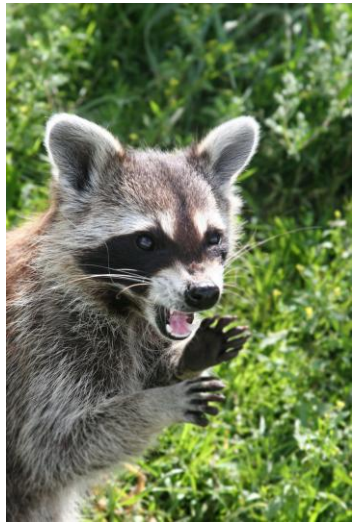
Sumário

- Redes Neuronais Convolucionais
  - Conceito – os neurónios como unidades que realizam a convolução
  - Arquitetura de uma rede convolucional
    - Camadas convolucionais
    - Camadas de subamostragem (*pooling*)
    - Camadas densas (*fully-connected*)
  - Implementação no Tensorflow
  - *Overfitting* e formas de o reduzir:
    - *Dropout*
    - *Data augmentation*
  - Exemplos

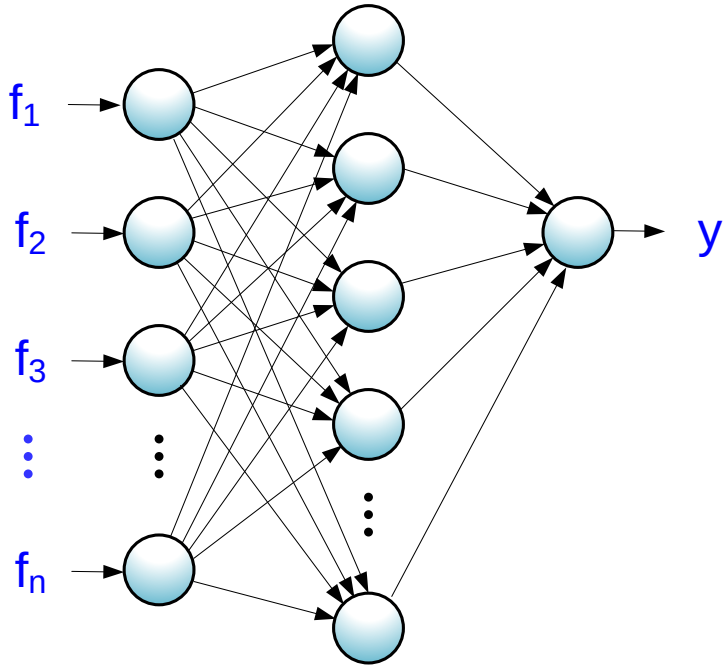
iscte
TECNOLOGIAS
E ARQUITETURA

Redes Neuronais Convolucionais (CNN)

Classificação de imagens usando NNs

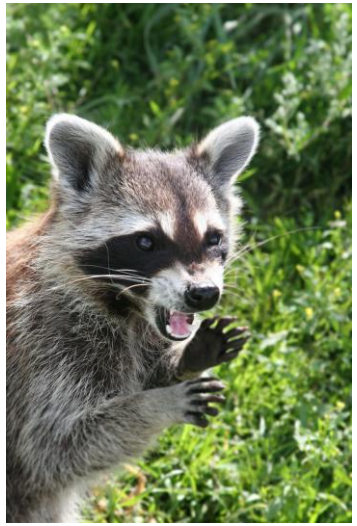


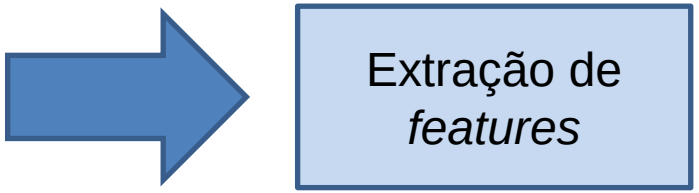
???

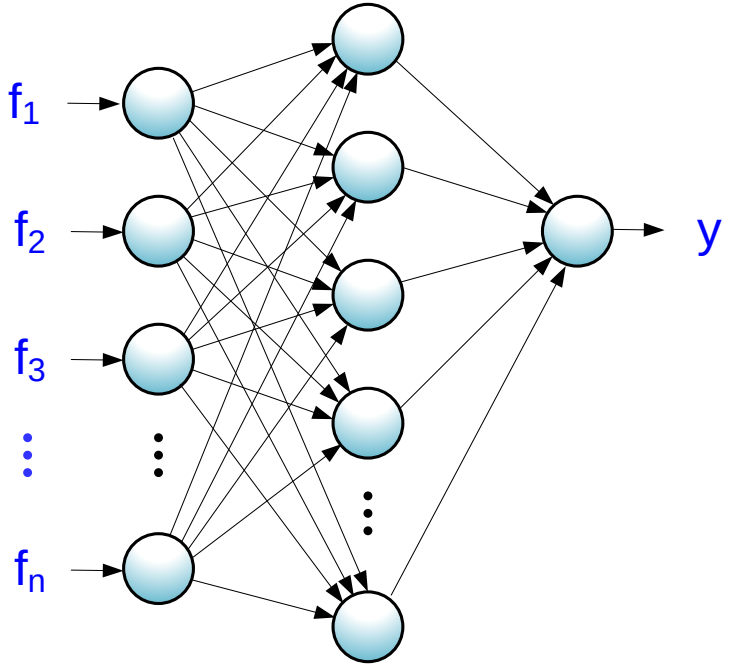


- Como ligar uma imagem a uma rede neuronal?
  - Ligar diretamente, onde cada entrada seriam valores de pixel?
  - Problemas: muitos parâmetros (pesos) para otimizar, não se tira partido da estrutura espacial das imagens…

Classificação de imagens usando NNs



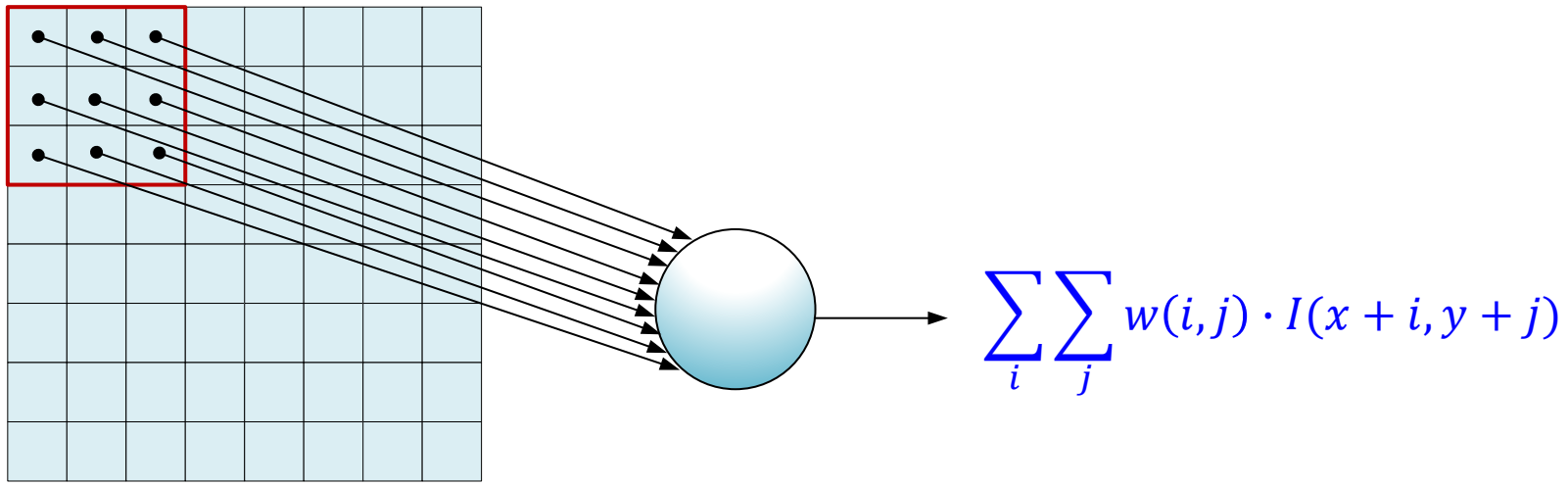




Abordagem “clássica”: extrair atributos (*features*) da imagem

- A rede produz as predições com base em atributos das imagens
- Problemas: a extração de *features* varia muito de problema para problema; é fácil extrair *features* de “baixo nível” (cores, contornos, texturas, etc.) mas é difícil extrair *features* de alto nível relacionadas com a estrutura dos objetos

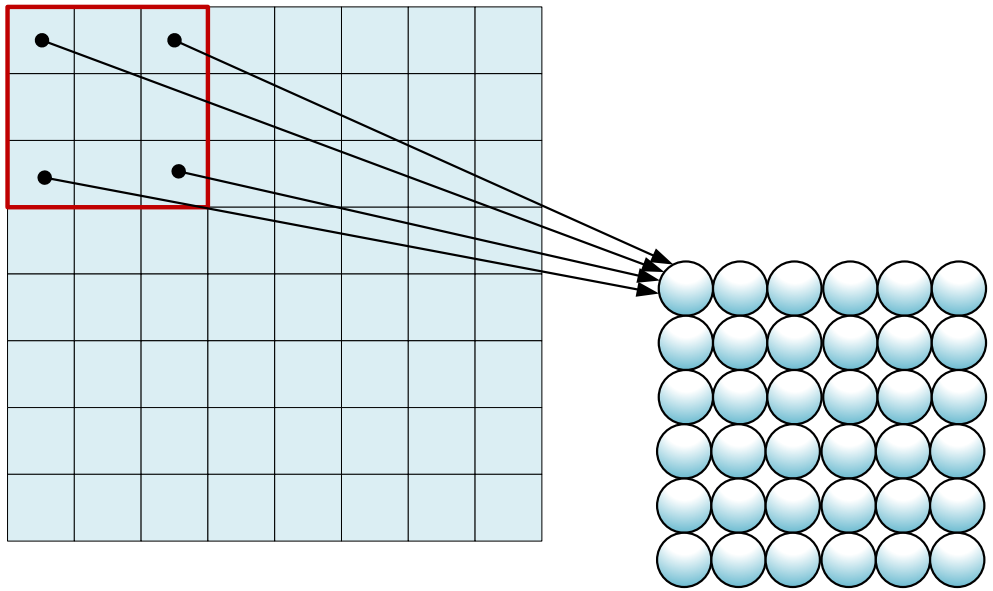
Ideia: Adequar a arquitetura da rede às imagens

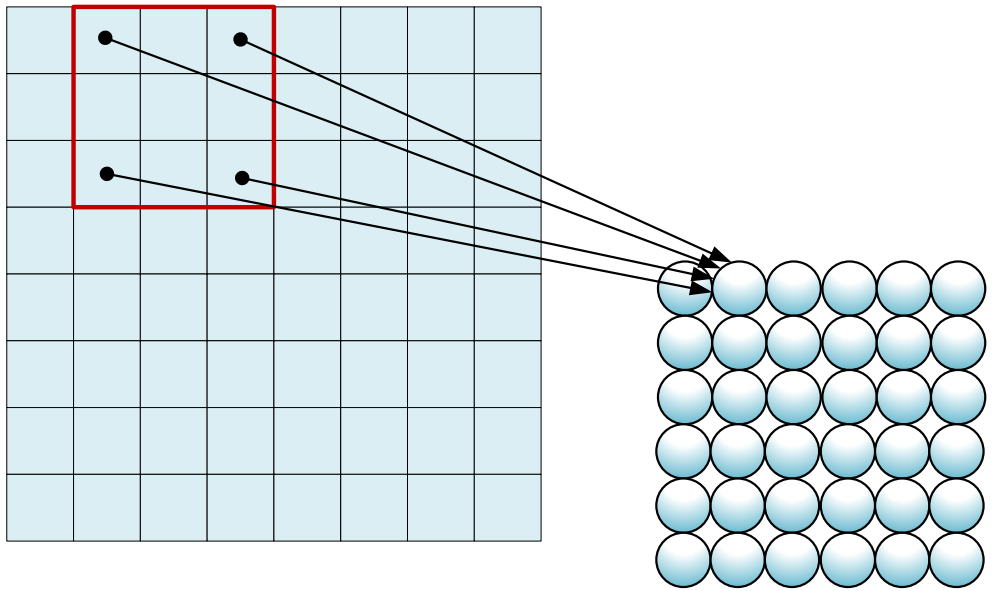


Ideia principal

- Tirar partido da estrutura espacial presente nas imagens
- Ligar os pixels a neurónios, organizando-os por blocos (que se podem sobrepor)
- Os blocos deverão ter dimensões pequenas face à imagem
  - *e.g.*, 3x3, 5x5

Arquitetura adequada para imagens



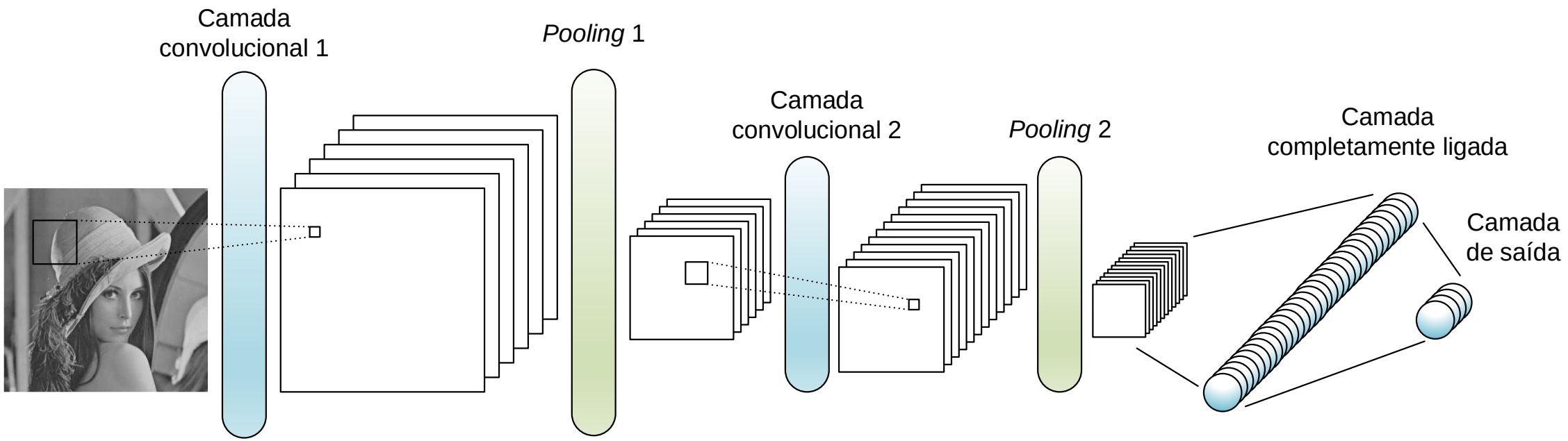


- Cada bloco da imagem é ligado a um neurónio; a camada contendo os neurónios (assim como os valores das suas saídas) pode ser vista como uma matriz
- Os neurónios que estão na mesma camada partilham entre si os mesmos pesos nas ligações
- Na prática temos uma **operação de convolução** com um filtro, onde os coeficientes do filtro correspondem aos pesos das ligações aos neurónios

Redes CNN – princípio de funcionamento

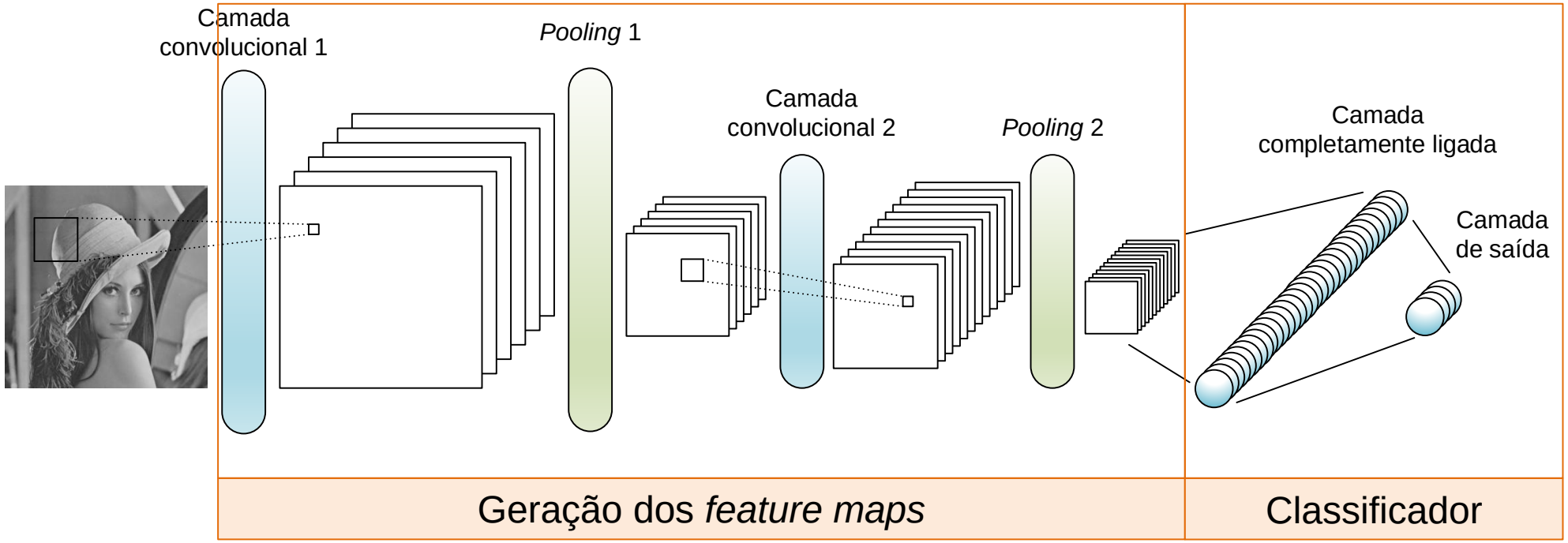
- A cada um dos blocos de imagem, aplica-se o mesmo conjunto de pesos
  - O resultado produzido é na prática a convolução de uma imagem com um **filtro**
  - Os pesos das ligações dos neurónios são os coeficientes do filtro
  - O resultado pode ser visto como um **mapa de características** (***feature map***)
- Para se obterem diferentes *feature maps*, utilizam-se diferentes filtros
  - Tantos filtros quanto o número de *feature maps* que se pretendem gerar através da camada convolucional
- Os pesos dos filtros são obtidos automaticamente durante o treino da rede
  - Na prática isso significa que os próprios filtros são treinados de forma a produzirem *feature maps* relevantes para o problema em questão

Estrutura de uma CNN para classificação



- **Camadas convolucionais:** aplicam os filtros para se obterem os *feature maps*
  - Tipicamente seguidas de unidades de ativação **ReLU** que colocam a '0' os valores negativos
- ***Camadas de sub-amostragem (Pooling)***: reduzem a dimensão dos *feature maps*
- ***Camadas densas*** – idênticas à camada "escondida" numa rede neuronal clássica

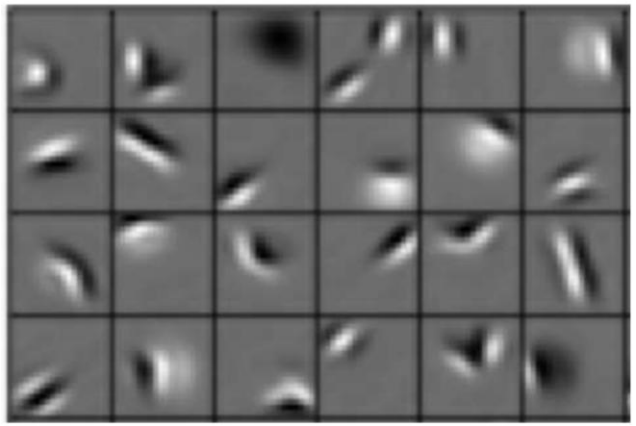
Estrutura de uma CNN para classificação



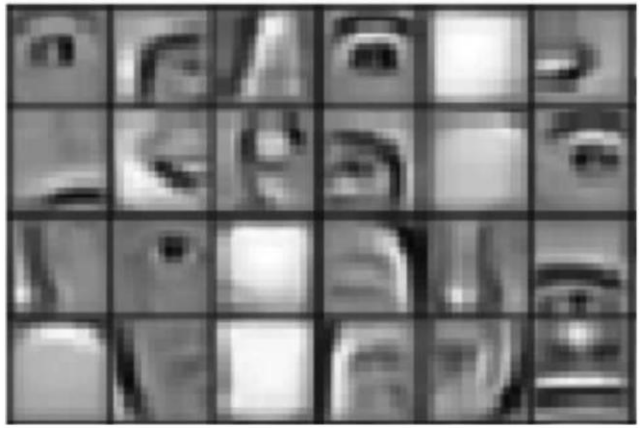
- A CNN pode ser vista como a junção de duas partes
  - Camadas convolucionais e de *pooling* – responsáveis por gerar os *feature maps*
  - As camadas completamente ligadas e de saída – que na prática implementam um classificador idêntico a uma rede neuronal clássica

Aprendizagem automática de *features*

O computador vai aprendendo uma hierarquia de *features*



Baixo nível
Manchas
Linhas
Cor



Nível médio
Olhos
Bocas
Narizes

Alto nível
Estrutura facial

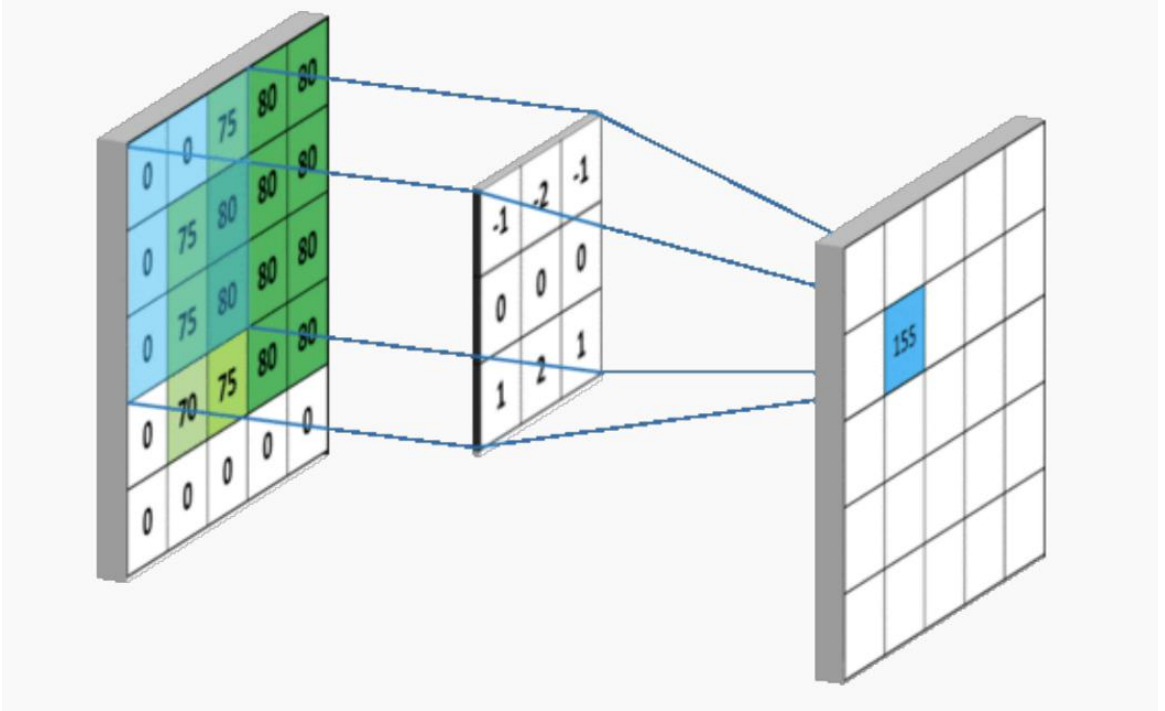
Camadas convolucional inicial

Última camada convolucional

Camadas convolucionais

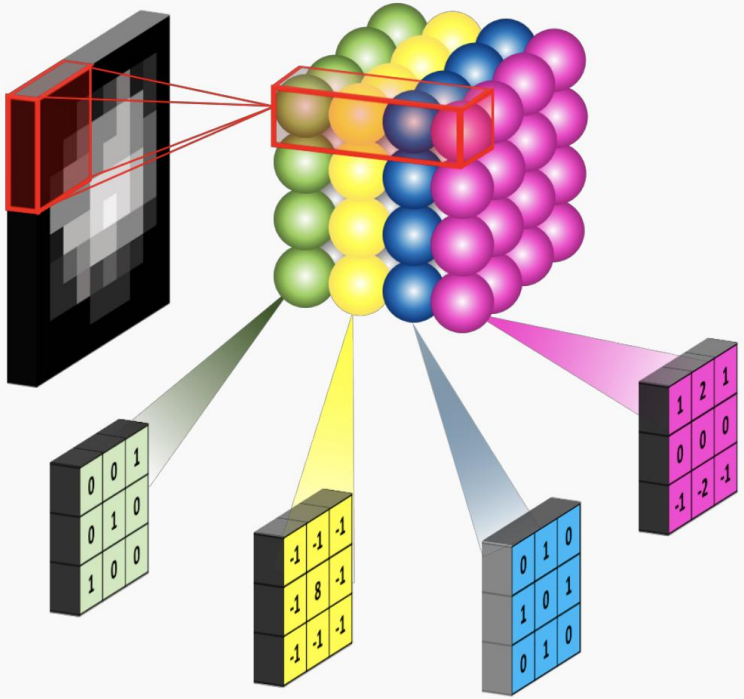
$$(I * F)(x,y) = \sum_{m=-a}^{a} \sum_{n=-b}^{b} I(x+m, y+n) F(m+a, n+b)$$

$$a = \left\lfloor \frac{F_W}{2} \right\rfloor, b = \left\lfloor \frac{F_H}{2} \right\rfloor$$

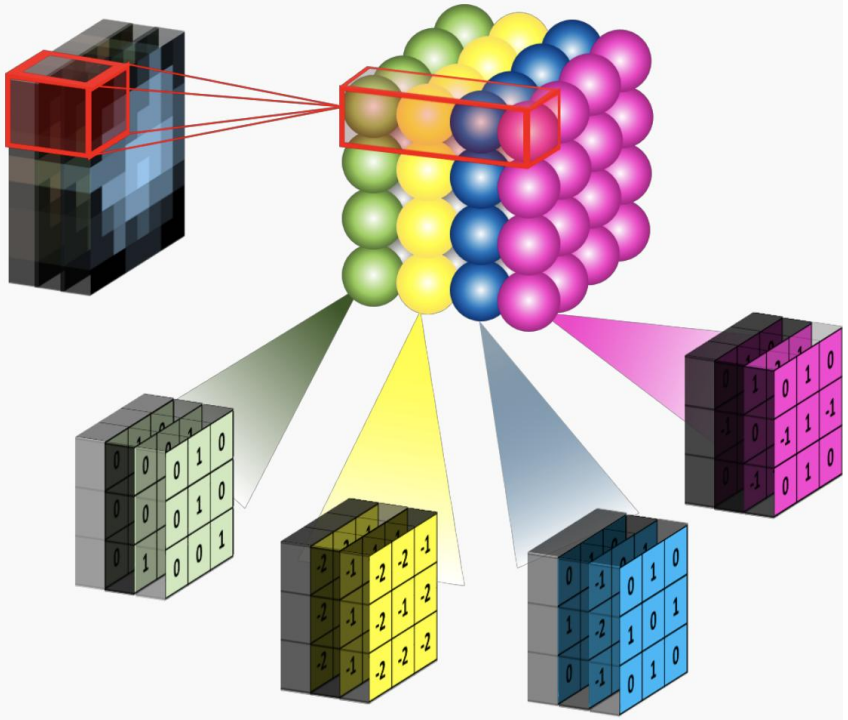


- Realizam convoluções com filtros
- Os coeficientes dos filtros são pesos aprendidos pela rede
- Parâmetros importantes: número (NF) e dimensões $F_W \times F_H$ dos filtros
- Produz NF matrizes, que são o resultado da convolução com cada um dos filtros

Camadas convolucionais



Convolutional layer with four 3x3 filters on a black and white image (just one channel)

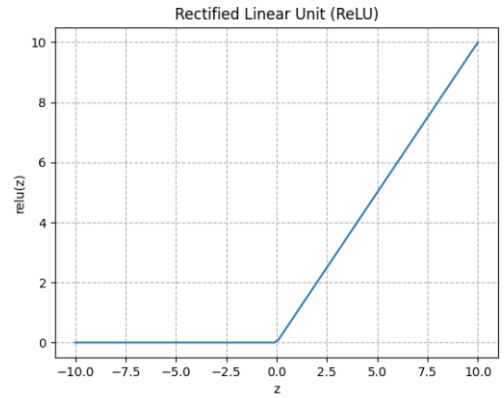


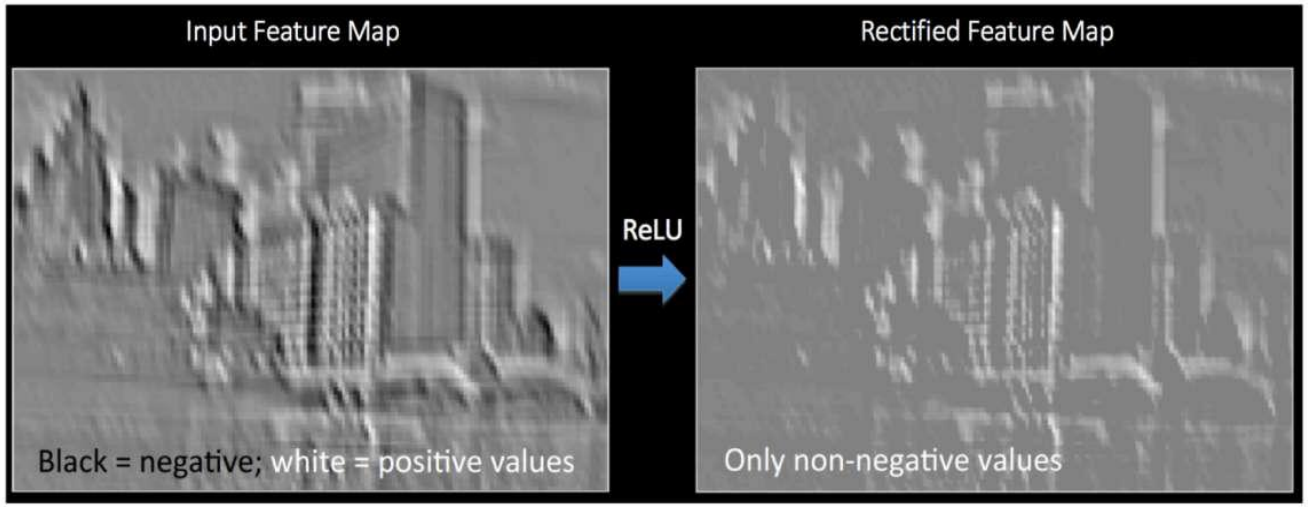
Convolutional layer with four 3x3 filters on an RGB image. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

Ativação ReLU

- ReLU – *Rectified Linear Unit*
- Aplica-se tipicamente à saída das camadas convolucionais
- O seu efeito é o de substituir valores negativos por 0's

$$ReLU(x) = \max(0, x)$$





Vantagens face à sigmoide:

- Computacionalmente mais eficiente
- Evita um problema conhecido por "*vanishing gradient*" – gradiente com valores próximos de 0 que atrasa ou impede a convergência dos processos de optimização

Camadas de subamostragem (Pooling)

10	0	0	0	0	1
5	2	2	5	3	0
3	5	3	8	3	0
8	9	1	0	0	8
0	0	0	0	0	1
1	3	0	0	4	1

Exemplo:
MaxPooling usando uma janela de 2 x 2

10	5	3
9	8	8
3	0	4

- Reduz a dimensionalidade – importante para diminuir o número de operações em camadas subsequentes
- Introduz alguma robustez em relação a pequenas translações nas imagens

Implementação no Tensorflow

```
model = keras.Sequential([
    layers.Input(shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 5, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation="softmax")
```

Argumentos das camadas convolucionais Conv2D:

- Número de filtros – nesta camada são 16
- Dimensão dos filtros – se for apenas um inteiro assume filtros quadrados – nesta camada são 5x5
- `padding` – acrescenta zeros à volta da matriz de entrada. Ao usar `'same'` preenche de maneira a que o resultado da convolução seja uma matriz com a mesma dimensão da matriz de entrada
- `activation` – função de ativação

iscte
TECNOLOGIAS
E ARQUITETURA

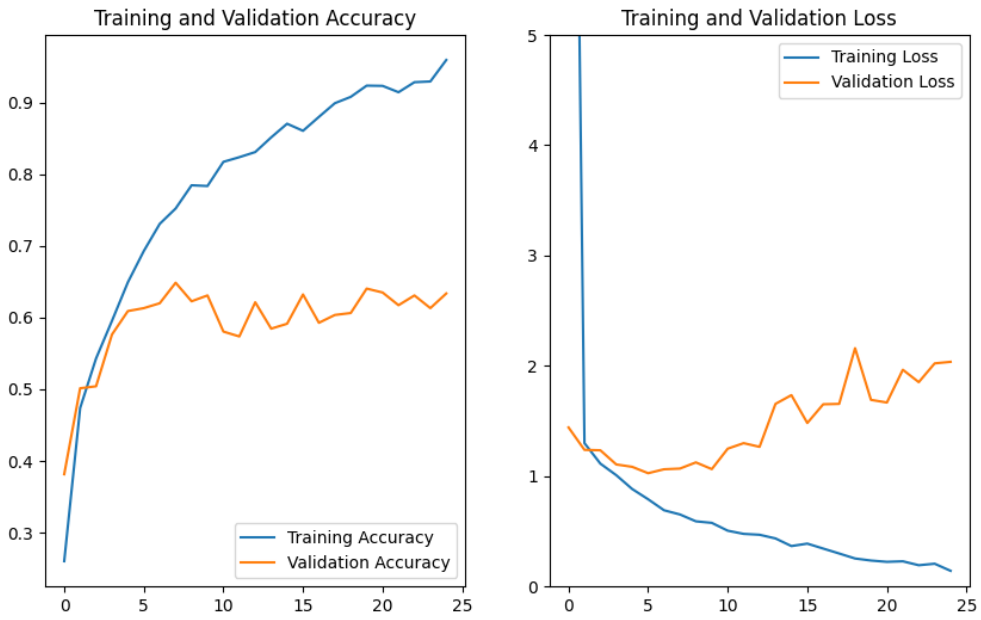
Overfitting e formas de o reduzir

Data Augmentation

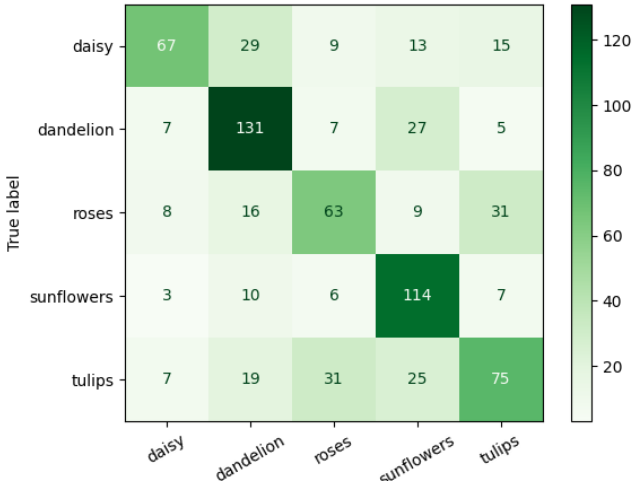
Dropout

Resultados Flower_Photos (I)

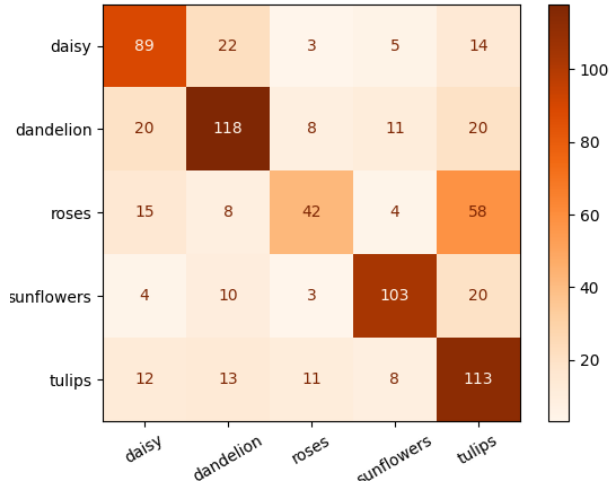
Resultados com uma CNN simples, longe de serem satisfatórios…



Modelo com menor loss (época 5)



Modelo da última época

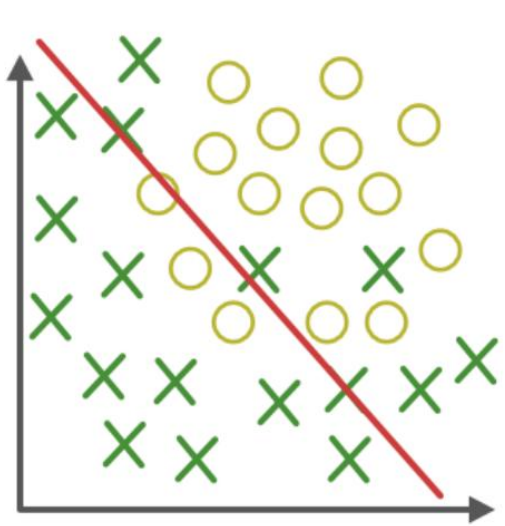


A rede aprende, mas não generaliza bem:
na última época, a *acc* do conjunto de treino é alta (+95%), mas a da validação é baixa (~60%)

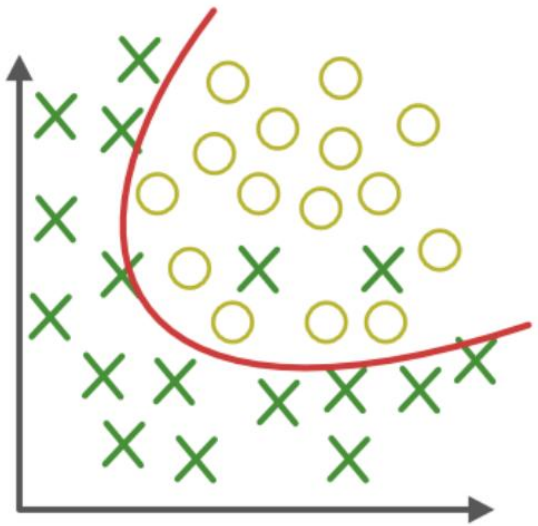
Está-se numa situação de ***overfitting***!

Overfitting (I)

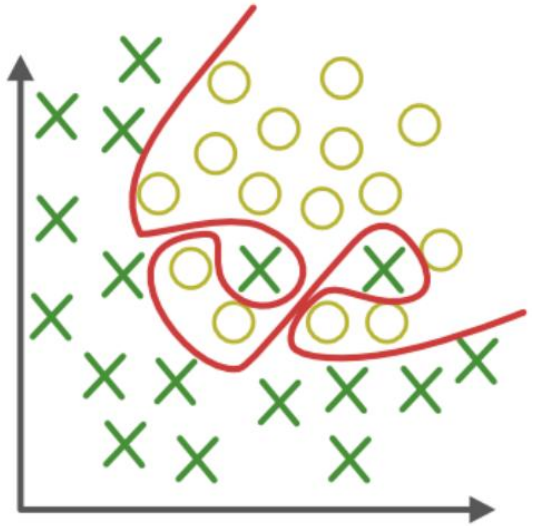
- **Sobre-ajuste (*overfitting*)** – situação em que a rede tem bom desempenho no conjunto de treino, mas não nos conjuntos de validação e teste
- Na prática aprende "particularidades" do conjunto de treino – isso faz com que não generalize o que aprendeu para dados que não tenham entrado no treino



Underfitting
(modelo demasiado simples)



Good fit
(modelo que generalizou bem)

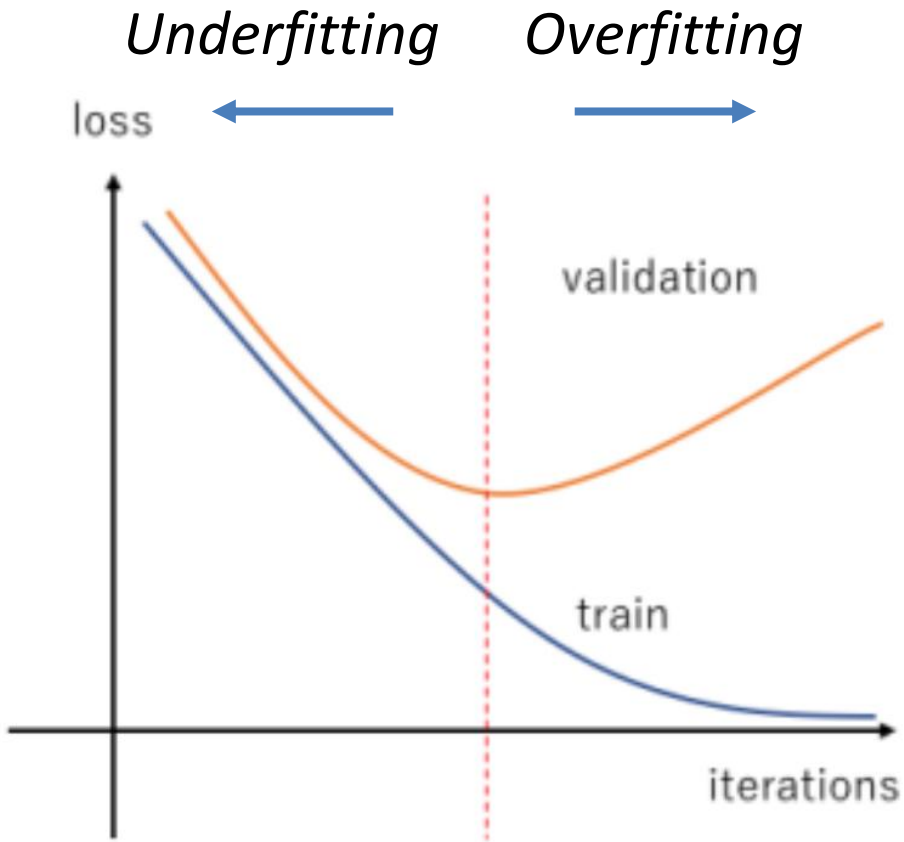


Overfitting
(ajustado ao conjunto de treino)

Overfitting (II)

O *overfitting* pode ser detetado durante o treino de uma rede neuronal observando a evolução da função de perda

Quando a função de perda no conjunto de validação deixa de decrescer não há vantagem em continuar o treino, pois a rede começa a adaptar-se em demasia ao conjunto de treino e pode-se entrar numa situação de *overfitting*



Formas de evitar *overfitting*

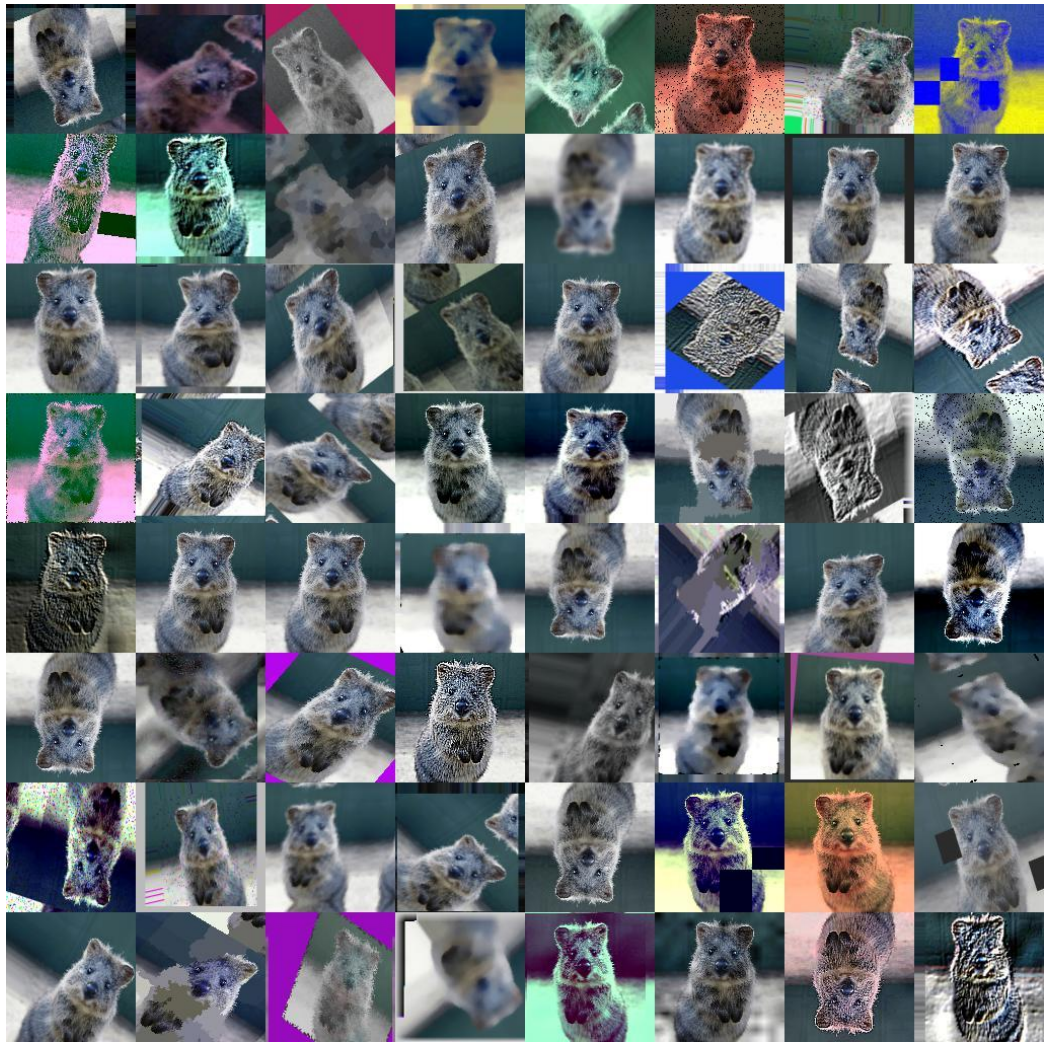
- Simplificar o modelo – modelos mais complexos do que deviam estão mais expostos ao *overfitting*
- Utilizar técnicas de regularização na função de perda
  - tipicamente introduzindo-se um termo na *loss* que penaliza o somatório dos valores absolutos dos pesos (ou o somatório dos quadrados dos pesos)
- Usar técnicas de ***data augmentation***
- Utilizar ***dropout*** durante o treino da rede

Data Augmentation

Consiste em gerar novas versões das imagens do conjunto de treino

- Zoom in / zoom out
- “Flips” (horizontal / vertical)
- Rotações
- Introdução de ruído
- …

Data augmentation



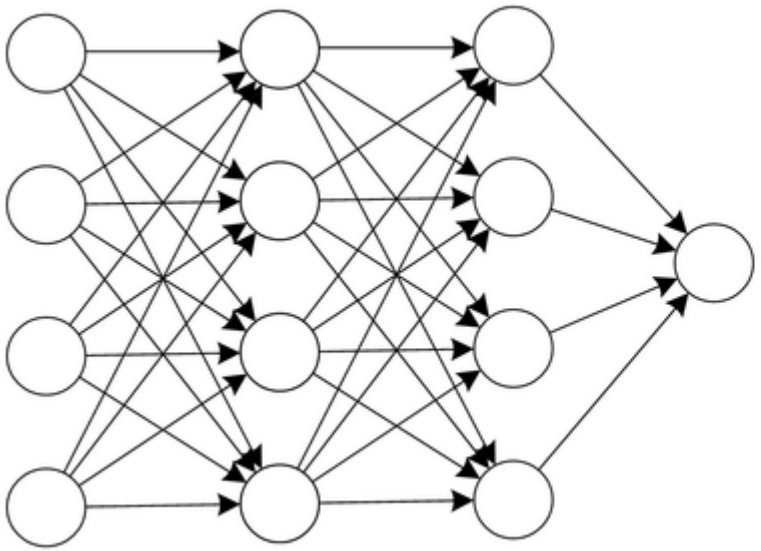
Dropout

Em cada iteração do treino da rede neuronal, escolhe-se aleatoriamente um conjunto de neurónios que vão ficar “desligados” da rede

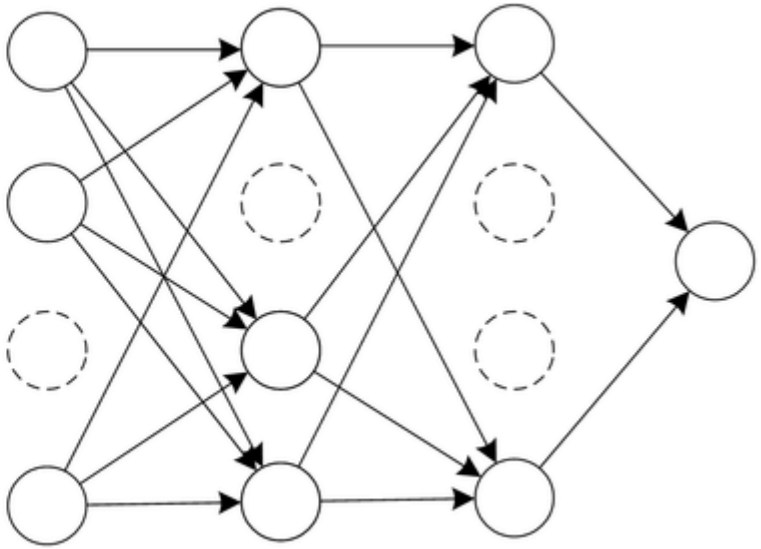
Durante o treino, escolhe-se um novo conjunto de neurónios “desligados” em cada iteração

Deste modo evita-se que algumas ligações se adaptem em demasia a eventuais particularidades do conjunto de treino

s/ *dropout*

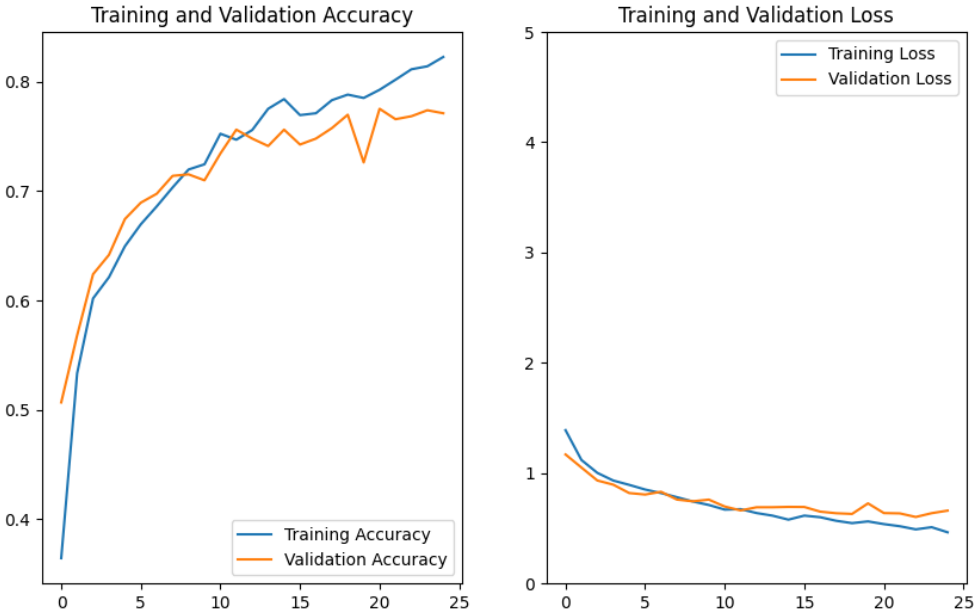


c/ *dropout*

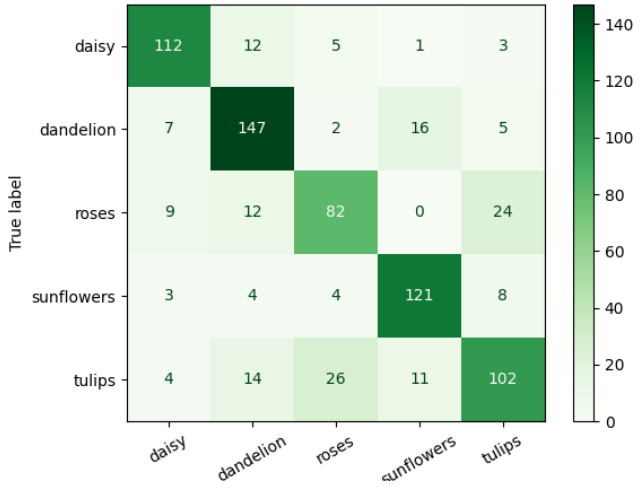


Resultados Flower_Photos (II)

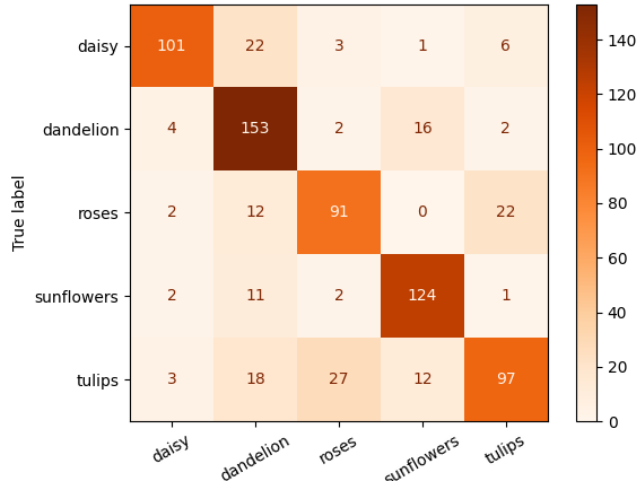
Utilização de *Data Augmentation* e *Dropout*



Modelo com menor *loss* (época 22)



Modelo da última época

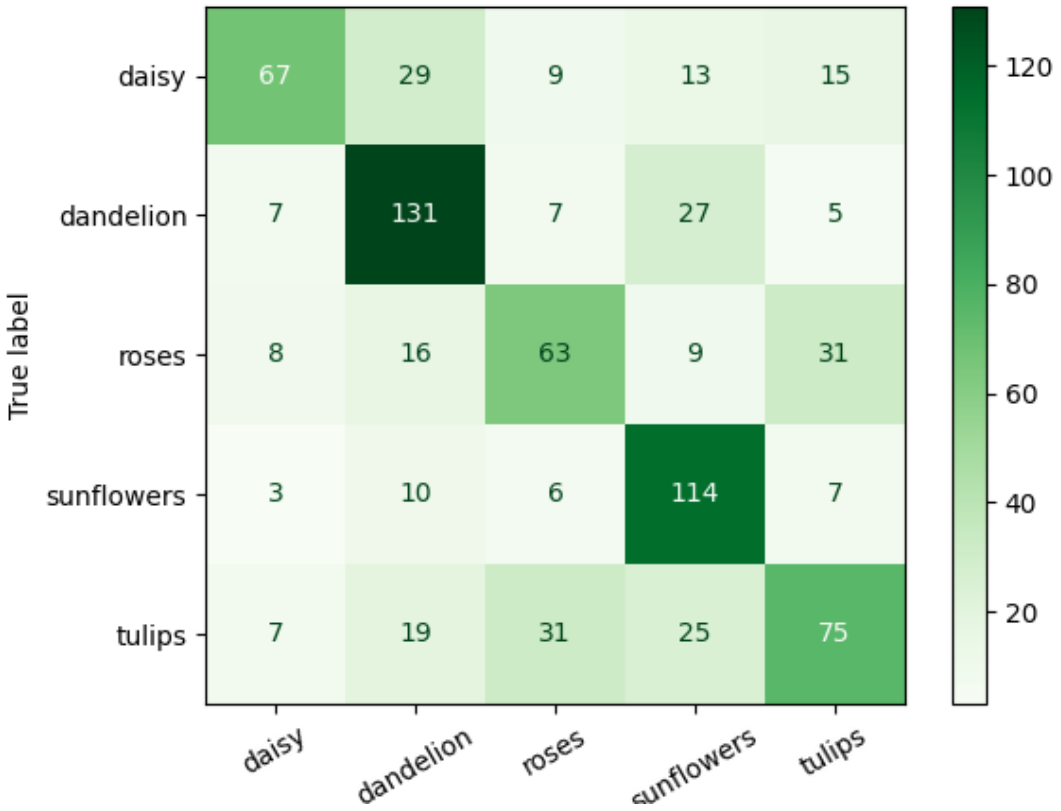


A rede generaliza melhor face ao caso anterior:
na última época, a *acc* do conjunto de treino é cerca de 81% e a da validação é cerca de 77%

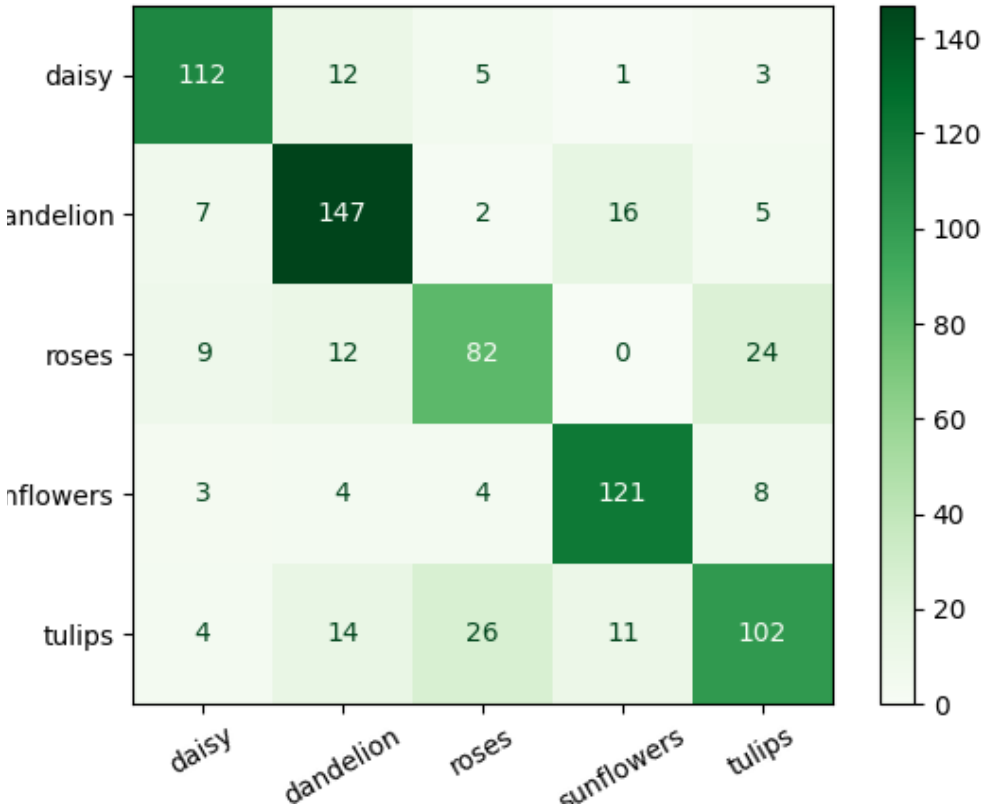
Não estão tão longe uma da outra como na primeira abordagem

Comparação

Versão mais simples do treino



Utilização de *Data Augmentation* e *Dropout*



Recursos

- Imagens
  - “Angry racoon”, https://www.freeimages.com/photo/angry-racoon-1548067
  - “Overfitting”. https://www.geeksforgeeks.org/regularization-in-machine-learning/
- Tutoriais tensorflow, https://www.tensorflow.org/tutorials
- M. Stewart, Simple Introduction to Convolutional Neural Networks https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac
- A. Soleimany and A. Amini, *Introduction to Deep Learning*, online course, MIT, 2021, http://introtodeeplearning.com/