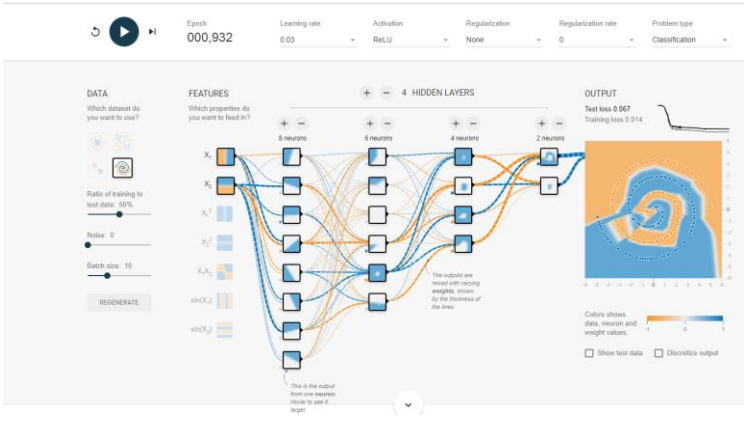
10
Redes de “Deep Learning”

IC 22/23

CPereira

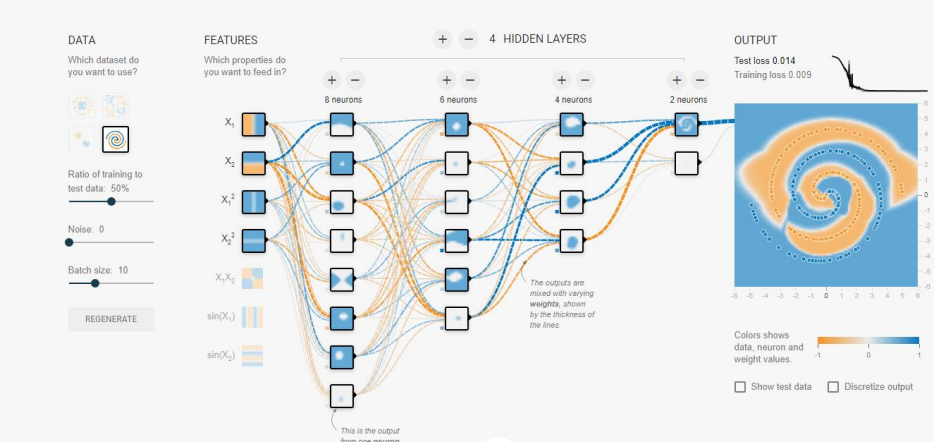
Aprendizagem de “features”

- Que propriedades devemos considerar? x1 e x2?
  - Quantas camadas e neurónios subsequentes?



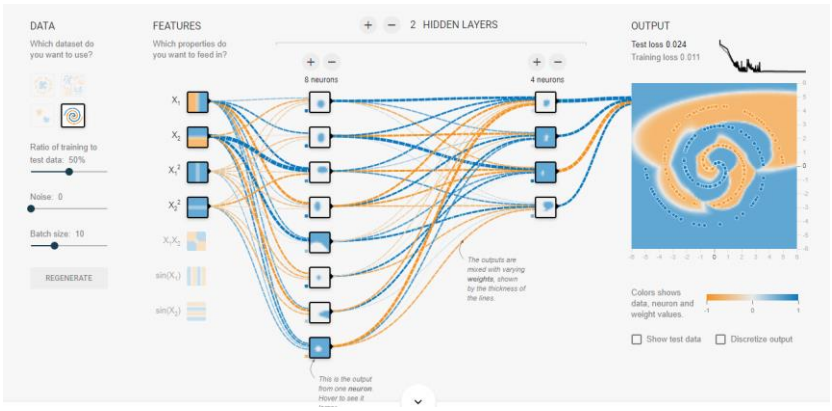
Aprendizagem de “features”

- E se considerarmos as *features* “x*x”?



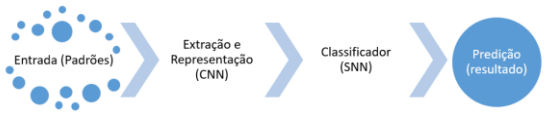
Aprendizagem de “features”

- ….
  - Compromisso, menos camadas…



Aprendizagem de “features”

- Aprendizagem automática de “features” (características)
  - Um primeiro bloco (várias camadas) para extração e representação das “features” do problema recorrendo habitualmente a redes convolucionais.
  - Após esta fase, recorre-se a uma rede de pouca profundidade (designada de “shallow network”) com aprendizagem supervisionada.

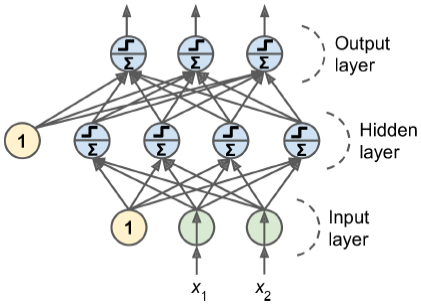


Redes com múltiplas camadas

- As designadas redes neuronais com poucas camadas – rasas ou “shallow” não conseguem representar a estrutura subjacente a padrões de treino complexos, tal como como uma imagem, texto, vídeo ou sinal áudio.
- Com base em estudos nas áreas de neurociência, as redes neuronais profundas (com muitas camadas) imitam de forma mais fidedigna o processo de visualização de imagens no córtex, sendo assim adequadas à automatização de tarefas “humanas”.

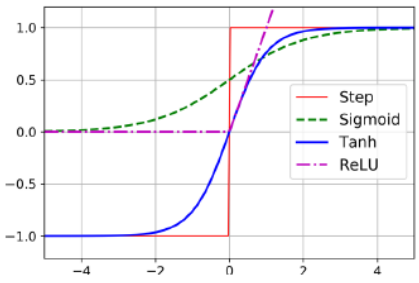
Redes “Shallow”

- Redes com pouca profundidade
  - Redes MLP com uma camada interna[1]
    - 2 entradas
    - Uma camada interna com 4 neurónios
    - 3 saídas

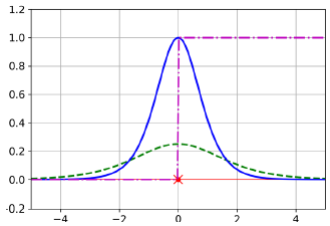


Redes “Shallow”

- …
  - Funções de ativação típicas [1]

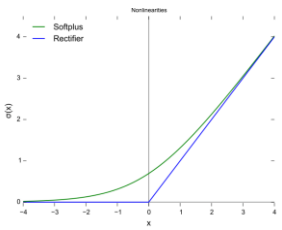


  - e derivadas



Redes "Shallow"

- …
  - Função RELU *ReLU(z) = max(0, z)*
    - *Funcional melhor do que função sigmoidal, apesar da analogia biológica.*
    - *Vantagens*
      - *Não tem um valor de saída máximo, o que ajuda a reduzir alguns problemas durante o Gradient Descent*
      - *Eficiente – cálculo mais rápido*
    - *Desvantagens*
      - *Não diferenciável em z=0*
      - *Derivada = 0 para z<0*



$$f(x) = x^+ = \max(0, x),$$

Redes "Shallow"

- …
  - *Função "softmax"*
    - *Quando precisamos de um neurónio por classe*
    - *todas as probabilidades estimadas estão entre 0 e 1 e que somam 1*

```
>>> import numpy as np
>>> a = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> np.exp(a) / np.sum(np.exp(a))
array([0.02364054, 0.06426166, 0.1746813, 0.474833, 0.02364054,
       0.06426166, 0.1746813])
```

    - normaliza a saída de uma rede para uma distribuição de probabilidade sobre as classes de saída previstas

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Redes MLP

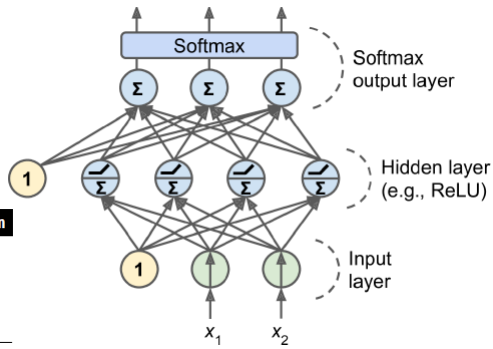
- …
  - Problemas de regressão – arquitetura típica [1]

Hyperparameter	Typical value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Redes MLP

- …
  - Problemas de Classificação

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy



- *Multilabel* – Várias classes numa instância.
  - Exemplo . Reconhecimento de imagem; 3 pessoas; “Maria”, “João” “Manuel”; numa imagem (instância) com a Maria e Manuel, o output poderá ser [1 0 1]
- *Multiclass* – Apenas identifica uma das classes numa instância

Redes Profundas

- Redes Profundas
  - Redes com muitas camadas
- Redes Densas ou totalmente conectadas
  - Quando todos os neurónios de uma camada estão conectados a todos os neurónios da camada anterior, a camada é designada de “totalmente conectada” ou camada “densa”.
- TensorFlow
  - O número excessivo de parâmetros torna incomportável o treino eficiente destas redes profundas num ambiente de computação tradicional.
  - A biblioteca “Tensorflow” foi criada para otimizar e distribuir processos de cálculo complexos, onde os problemas são representados em forma de grafos.
    - O princípio de funcionamento é habitualmente simples. Define-se, um gráfico de computação o tensorflow otimiza e distribui a sua execução
    - A organização eficiente permite a execução de código em múltiplos CPUs e GPUs.

Redes Profundas

- …
  - Keras
    - Quando o nível de complexidade pretendido corresponde à construção de uma rede neuronal o TensorFlow revela-se complexo, recorrendo-se então a APIs específicas, tal como o Keras.
    - A biblioteca Keras fornece uma camada de construção de redes neuronais profunda utilizando o TensorFlow como suporte e inclui vários modelos pré-definidos, nomeadamente para reconhecimento de imagem.
    - o Keras é provavelmente a forma mais expedita de criar redes profundas e complexas com um número muito reduzido de linhas de código.

Redes Profundas

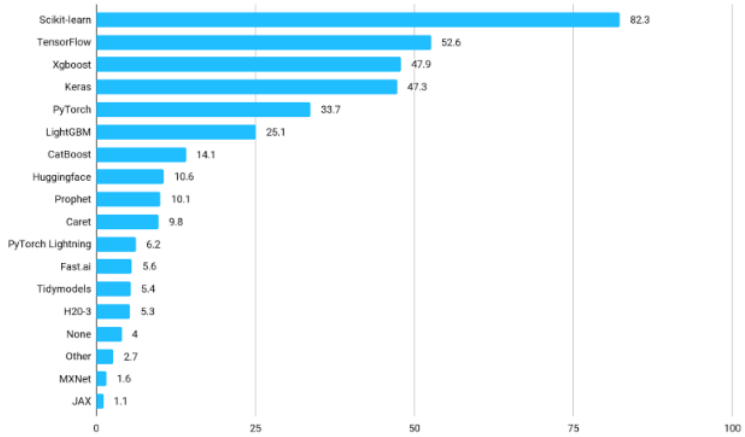
- …
  - O modelo “sequential” corresponde a uma pilha linear de camadas com arquiteturas e funções distintas.
    - A rede pode ser criada passando como argumento ao construtor uma lista de instâncias de diferentes camadas.
    - Exemplo de uma rede densa, totalmente conectada com 784 neurónios e função de ativação “relu”, seguida de uma outra rede totalmente conectada com 10 neurónios e função de ativação “softmax”:

```
[ ]: from keras.models import Sequential
     from keras.layers import Dense, Activation

     model = Sequential([
         Dense(32, input_shape=(784,)),Activation('relu'),
         Dense(10), Activation('softmax'),
     ])
```

Redes Profundas

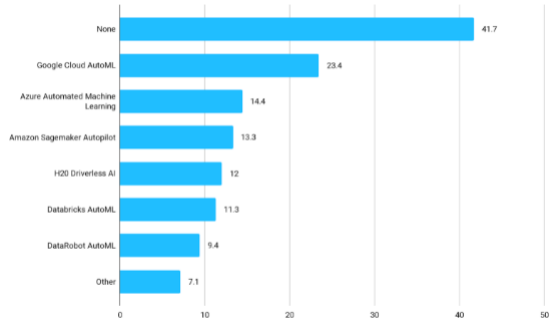
- …
  - Ferramentas

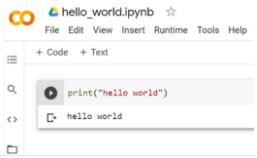


- Fonte: https://www.kaggle.com/kaggle-survey-2021

Redes Profundas

- …
  - Cloud

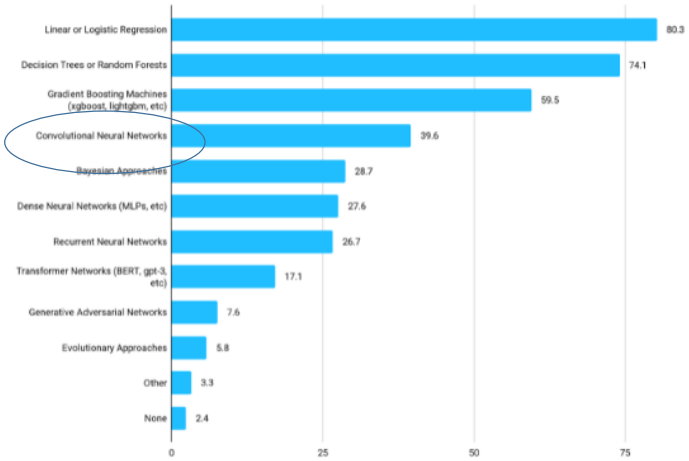




  - Google Colab

Redes Profundas

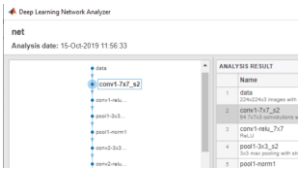
- …
  - Algoritmos

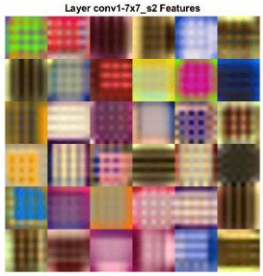


  - Fonte: https://www.kaggle.com/kaggle-survey-2021

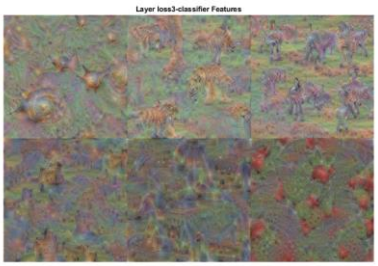
Redes Profundas

- ...
  - Net=googlenet [2]





.....

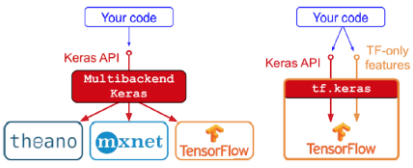


- L1 - as imagens contêm principalmente bordas e cores.
- L2 - filtros para esta camada detetam padrões mais complexos do que a primeira camada convolucional.
- Na última camada, as imagens geradas ativam fortemente as classes (zebra, castelo) selecionadas.

Exemplo

- Fashion Mnist [1]
  - 10 classes

  - Implementações de Keras
    - *multibackend*
    - *tf.keras*
      - import tensorflow as tf
      - from tensorflow import keras



Exemplo

- …

```
import tensorflow as tf
from tensorflow import keras

#dataset de treino e teste
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()

# 60000 imagens para treino 28*28 de 0 a 255
X_train_full.shape

#normaliza para 0 a 1 e forma dataset de validação
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

#define classes
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
#exemplo de coat
class_names[y_train[0]]

#define modelo
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

Exemplo

- …
  - Características do modelo:
    - pilha única de camadas conectadas sequencialmente:
      - (1) Flatten Layer - converter cada imagem de entrada num vetor unidimensional
      - (2) Dense hidden layer - 300 neurónios, função ‘relu’
      - (3) Dense hidden layer - 100 neurónios, função ‘relu’
      - (4) Dense layer - 10 neurónios, função ‘softmax’ (saídas)

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

    - Número de parâmetros = 235500+30100+1010 = 266610!!
      - Isso dá ao modelo capacidade para se ajustar aos dados de treino, mas também significa que o modelo corre o risco de “overfitting”

Exemplo

- …
  - Depois de criar o modelo
    - Chamar o método “compile()” para especificar a função de custo e a função de otimização. Opcionalmente, pode especificar-se uma lista de métricas a calcular durante o treino.
      - https://keras.io/api/optimizers/
      - https://keras.io/api/losses/
      - https://keras.io/api/metrics/

```
model.compile(loss="sparse_categorical_crossentropy",
optimizer="sgd",
metrics=["accuracy"])
```

    - Treinar o modelo invocando o método “fit()”

```
history=model.fit(X_train, y_train, epochs=30,
          validation_data=(X_valid, y_valid))
```

      - Em vez de passar um conjunto de validação pode indicar-se o “validation_split” para a proporção do conjunto de treino a usar para validação.
      - Se o desempenho no conjunto de treino é muito superior ao do conjunto de validação temos uma situação de “overfitting”
      - https://keras.io/api/models/model_training_apis/#fit-method

Exemplo

- …
  - Avaliação de Desempenho
    - O método fit() retorna um objeto “history”, contendo os parâmetros de treino (history.history) a lista de épocas (history.epoch) e um dicionário (history.history) contendo as métricas de avaliação.
    - Se o desempenho não for satisfatório, voltar atrás e ajustar os hiperparâmetros, nomeadamente:
      - Ajustar taxa de aprendizagem.
      - Tentar outro otimizador
      - ajustar hiperparâmetros do modelo como o número de camadas, o número de neurónios por camada e funções de ativação.
      - reajustar a taxa de aprendizagem após alterar qualquer hiperparâmetro.
  - Se o desempenho no treino é satisfatório, avaliar o desempenho no conjunto independente de teste para avaliar o erro de generalização:

```
model.evaluate(X_test, y_test)
```

    - https://keras.io/api/models/model_training_apis/

Exemplo

- …
  - Caso o erro de generalização seja aceitável, podemos iniciar o “deployment” do nosso modelo:

```
model.save("myModel")
```

  - Podemos usar o método predict() para fazer previsões para novas instâncias.
    - Faltando novos exemplos, neste caso, usamos três instâncias do conjunto de teste:

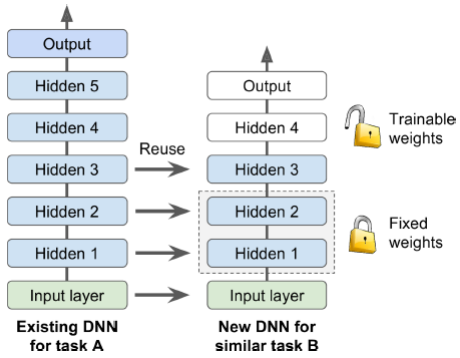
```
model = keras.models.load_model("myModel")

X_new = X_test[:3]
y_new = y_test[:3]

y_proba = model.predict(X_new)
y_pred = model.predict_classes(X_new)
```

Transferência

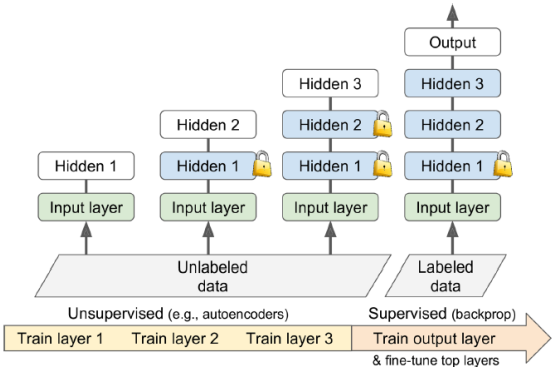
- Reutilização de modelos treinados [1]




```
model_A = keras.models.load_model("my_model_A.h5")
model_B_on_A = keras.models.Sequential(model_A.layers[:-1])
model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))
```

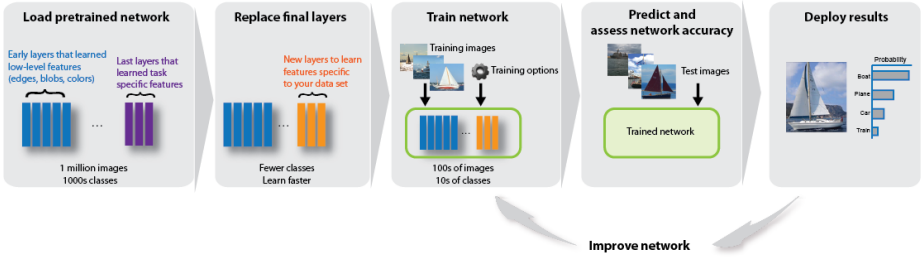
Transferência

- Pré-treino não supervisionado
  - Habitualmente, em tarefa complexas, podem existir muitas instâncias, contudo apenas uma pequena percentagem se encontra catalogada.
    - Nesse caso podemos usar duas fases, treino não supervisionado seguido de treino supervisionado transferindo o modelo não supervisionado [1].



Transferência

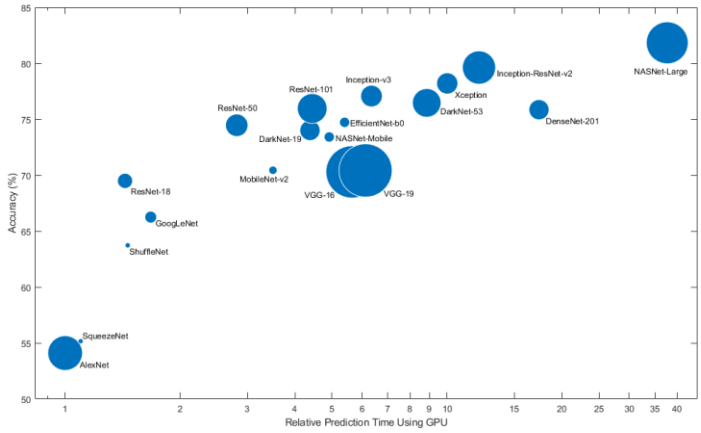
- Transferência de modelos – Fluxo [2]



- https://www.mathworks.com/help/deeplearning/ref/deepnetworkdesigner-app.html

Transferência

- …

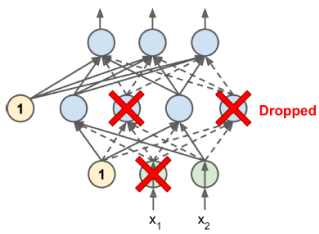


- Accuracy [2] para base de dados de imagens “ImageNet” , https://www.image-net.org/

Técnicas de regularização

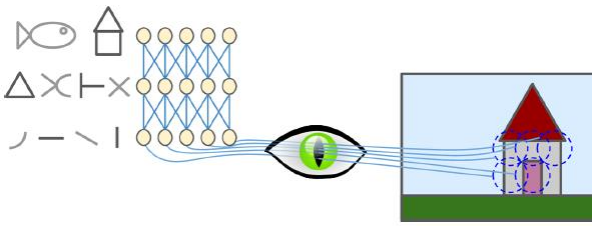
- Dropout
  - técnicas de regularização mais popular para redes profundas.
    - Em cada etapa do treino, cada neurónio (incluindo as entradas e excluindo obviamente os neurónios de saída) têm uma probabilidade “p” (habitualmente entre 10 e 50%) de ser temporariamente “descartado”, o que significa que será totalmente ignorado durante apenas essa dase de treino, mas pode estar ativo durante a próxima etapa.
    - Após o treino, todos os neurónios são considerados!

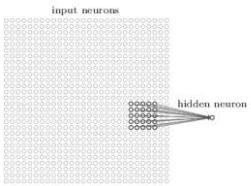
```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(300, activation="elu", kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(100, activation="elu", kernel_initializer="he_normal"),
    keras.layers.Dropout(rate=0.2),
    keras.layers.Dense(10, activation="softmax")
])
```



Redes Neuronais Convolucionais

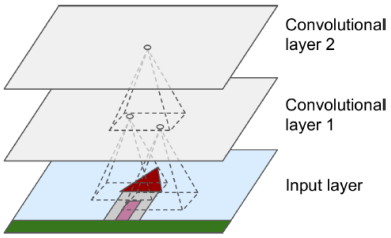
- Redes convolucionais CNN
  - Surgiram do estudo da visão no córtex cerebral, onde se constata que [1]:
    - Muitos neurónios possuem um pequeno **campo recetivo local**, o que significa que reagem apenas ao estímulos localizados numa região limitada do campo visual.
    - Mesmo dentro do mesmo campo recetivo local, alguns reagem **apenas a linhas horizontais**, enquanto outros reagem apenas a linhas verticais ou com orientações diferentes.
    - Alguns neurónios têm campos recetivos maiores e reagem a mais padrões mais complexos, que são **combinações dos padrões de nível inferior**.





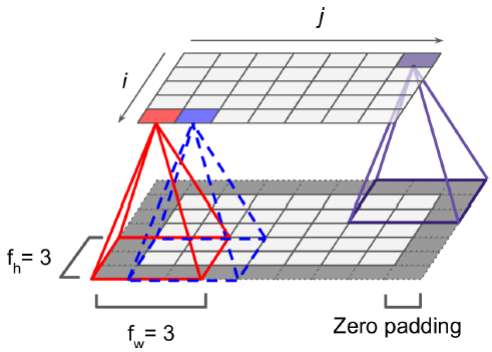
Redes Neuronais Convolucionais

- …
  - Camadas de convolução
    - Esta arquitetura permite que a rede se concentre em pequenas “features” de baixo nível na primeira camada oculta e, em seguida, combine-os em “features” mais complexas na próximo camada oculta, e assim sucessivamente…



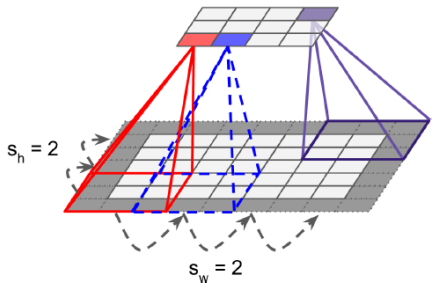
Redes Neuronais Convolucionais

- …
  - Exemplo
    - Filtros de 3*3
    - Slide de 1
    - Preenchimento de zeros
      - faz com que uma camada superior tenha a mesma altura e largura da camada anterior



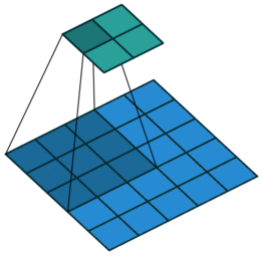
Redes Neuronais Convolucionais

- …
  - Redução de dimensão
    - Também é possível conectar uma camada a uma camada seguinte menor espaçando os campos recetivos (slide). O que permite reduzir a complexidade computacional.
      - Exemplo com espaçamento de dois nas duas dimensões, “slide”=2



Redes CNN

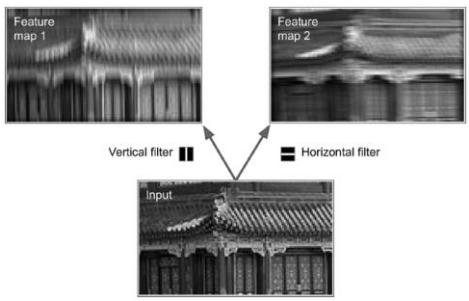
- ...
  - [2]



- https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html?searchHighlight=convolutional%20neural%20network&s_tid=doc_srchtitle

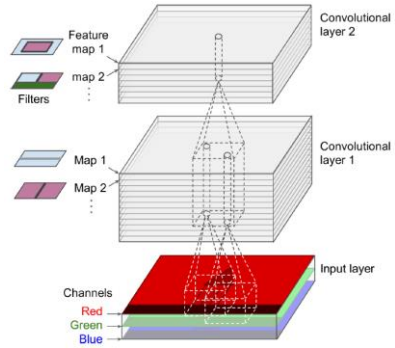
Redes Neuronais Convolucionais

- ...
  - As linhas brancas verticais (filtro vertical) ou horizontais (filtro horizontal) são realçadas enquanto o resto é desfocado.
  - uma camada usando o mesmo filtro produz um mapa de características (feature map), que destaca áreas de uma imagem.
  - O mais interessante é que estes "filtros" podem ser "aprendidos" – determinados pelo algoritmo, não necessitam de ser especificados previamente.



Redes Neuronais Convolucionais

- …
  - Podemos considerar vários filtros por "camada"
    - Assim as camadas são tridimensionais - conjuntos de feature maps.



Saída do neurónio da linha i, coluna j do mapa k

Bias (por feature map k)

Dimensão do filtro

Peso de ligação

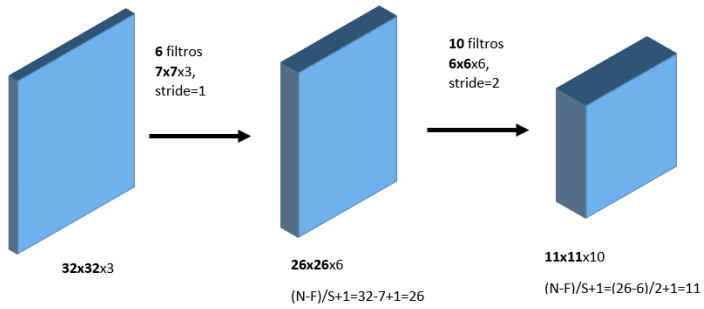
$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

Stride horizontal - h ou vertical - w

Saída do neurónio da camada anterior

Redes Neuronais Convolucionais

- …



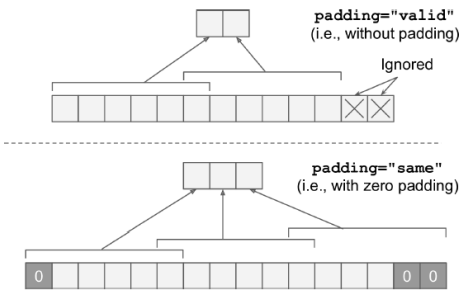
Redes Neuronais Convolucionais

- Implementação
  - Exemplo de uma camada de convolução 2D, 32 filtros 3×3, stride =1, "same" padding, função ReLU:

```
conv = keras.layers.Conv2D(filters=32, kernel_size=3, strides=1,
padding="same", activation="relu")
```

Redes Neuronais Convolucionais

- …
  - "Same" vs "Valid" padding [1]



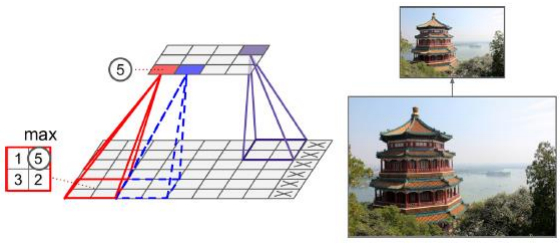
  - 13 entradas (1D), filtro de largura 6, stride=5

Redes Neuronais Convolucionais

- Pooling
  - Tem como objetivo reduzir a dimensão da imagem de entrada e restantes "features maps"
    - Reduzir a carga computacional,
    - Reduz o uso de memória,
    - Reduz o número de parâmetros (limitando assim o risco de overfitting).
  - Assim como nas camadas convolucionais, cada neurónio numa camada de pooling:
    - está conectado a um número limitado de neurónio na camada anterior, localizado dentro de um pequeno campo recetivo local.
      - Também é necessário definir o tamanho do "receptive fied", stride e padding;
      - No entanto, um neurónio da camada de pooling não tem pesos associados – apenas agrega as entradas, calculado o valor máximo (MaxPooling) ou a média.

Redes Neuronais Convolucionais

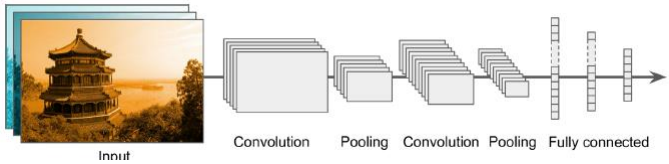
- …
  - Exemplo
    - 2×2 *pooling kernel, MaxPooling*, stride de 2, sem padding
      - Neste caso, reduz o tamanho da imagem




```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

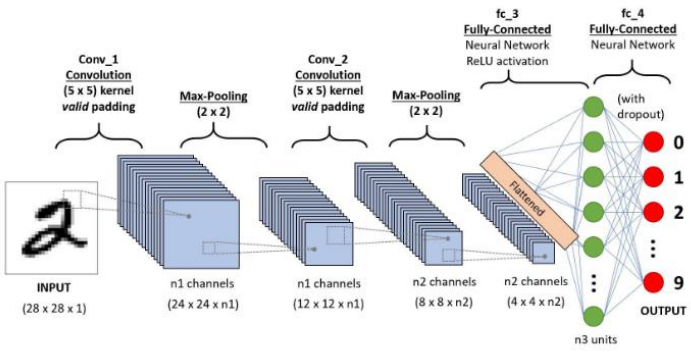
Redes CNN - Arquiteturas

- Arquitetura tradicional
  - As arquiteturas típicas empilham algumas camadas convolucionais (cada uma geralmente seguida por uma camada ReLU), em seguida, uma camada de Pooling, a seguir, outras camadas convolucionais + ReLU, a seguir outra camada de pooling, e assim sucessivamente [1]
    - A imagem fica habitualmente cada vez de menor dimensão à medida que avança pela rede, mas também fica cada vez mais profunda, ou seja, com mais "feature maps", devido ao processo de convolução.



Arquiteturas

- …
  - Modelo Típico



Fonte – "towardsdatascience.com"

Arquiteturas

- Exemplo - Fashion Mnist [1]
  - A primeira camada usa 64 filtros 7×7, sem stride, imagens de 28×28 pixels, com um único canal de cor (ou seja, tons de cinza).
  - Segue-se uma camada de Maxpooling de 2 - divide cada dimensão espacial por um fator de 2.
  - Em seguida, repetimos a seguinte estrutura duas vezes: duas camadas convolucionais seguidas por uma camada de pooling (o número de repetições é um hiperparâmetro a ajustar).
  - Em seguida define-se a rede densa - totalmente conectada, neste caso duas camadas internas e dropout.

```
"""
CNN
"""

model = keras.models.Sequential([
keras.layers.Conv2D(64, 7, activation="relu", padding="same",
input_shape=[28, 28, 1]),
keras.layers.MaxPooling2D(2),
keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
keras.layers.MaxPooling2D(2),
keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
keras.layers.MaxPooling2D(2),
keras.layers.Flatten(),
keras.layers.Dense(128, activation="relu"),
keras.layers.Dropout(0.5),
keras.layers.Dense(64, activation="relu"),
keras.layers.Dropout(0.5),
keras.layers.Dense(10, activation="softmax")
])
```

Arquiteturas

- …
  - Observações
    - Para a rede de convolução:
      - O número de filtros aumenta à medida que “subimos” na rede em direção à camada de saída (64, 128, 256).
      - Procedimento comum, uma vez que o número de ”features” de entrada - baixo nível é frequentemente bastante baixo (linhas horizontais, verticais, círculos, etc …), No nível seguinte devem considerar-se muitas formas de combinar estas “features” elementares.
      - Habitualmente dobra-se o número de filtros após cada camada de pooling – dado que uma camada de pooling divide cada dimensão espacial por um fator de 2, não corremos o risco de explodir o número de parâmetros.
    - Para a rede densa:
      - composta por duas camadas densas ocultas e uma camada densa de saída.
      - Deve-se nivelar as suas entradas, uma vez que uma rede densa espera uma matriz de uma dimensão por cada instância.
      - Adicionou-se neste caso duas camadas de “dropout”, com uma taxa de 50%, para reduzir o rico de overfitting.

Arquiteturas

- Outras Arquiteturas
  - LeNet-5
    - A arquitetura CNN mais conhecida
      - Rede para reconhecimento de caracteres manuscritos que designaram de “LeNet”. A arquitetura foi aplicada com bastante sucesso ao repositório “Mnist” (http://yann.lecun.com/exdb/lenet/).
      - As imagens MNIST têm 28×28 pixels, mas neste caso são preenchidas com zeros até 32×32 pixels e normalizadas antes de serem fornecidas à rede.

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

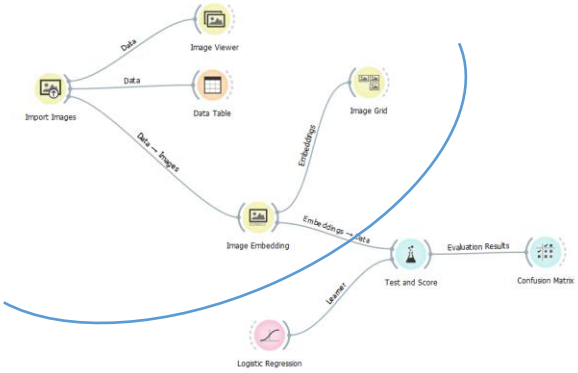
Arquiteturas

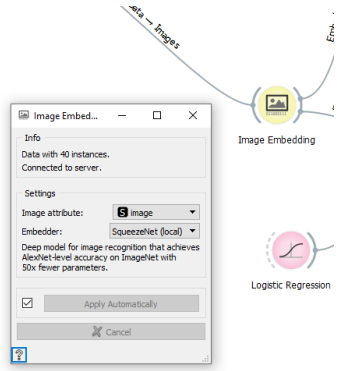
- …
  - AlexNet
    - Venceu o desafio ImageNet ILSVRC de 2012
    - Dropout de 50% para o treino para das camadas F9 e F10
    - Realizaram aumento de dados “data augmentation” deslocando aleatoriamente as imagens de treino por vários deslocamentos e mudar as condições de iluminação

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

Redes CNN

- Orange





- Modelos disponíveis treinados com "ImageNet":
  - SqueezeNet: Modelo simples mas rápido.
  - Inception v3: modelo da Google.
  - VGG-16: modelo de 16 camadas.
  - VGG-19: modelo de 19 camadas.

Referências

- *[1] Géron, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. "O'Reilly Media, Inc.", 2017.*
- *[2] https://www.mathworks.com*
- *https://www.tensorflow.org/*
- *https://keras.io/*
- *https://orange.biolab.si/*