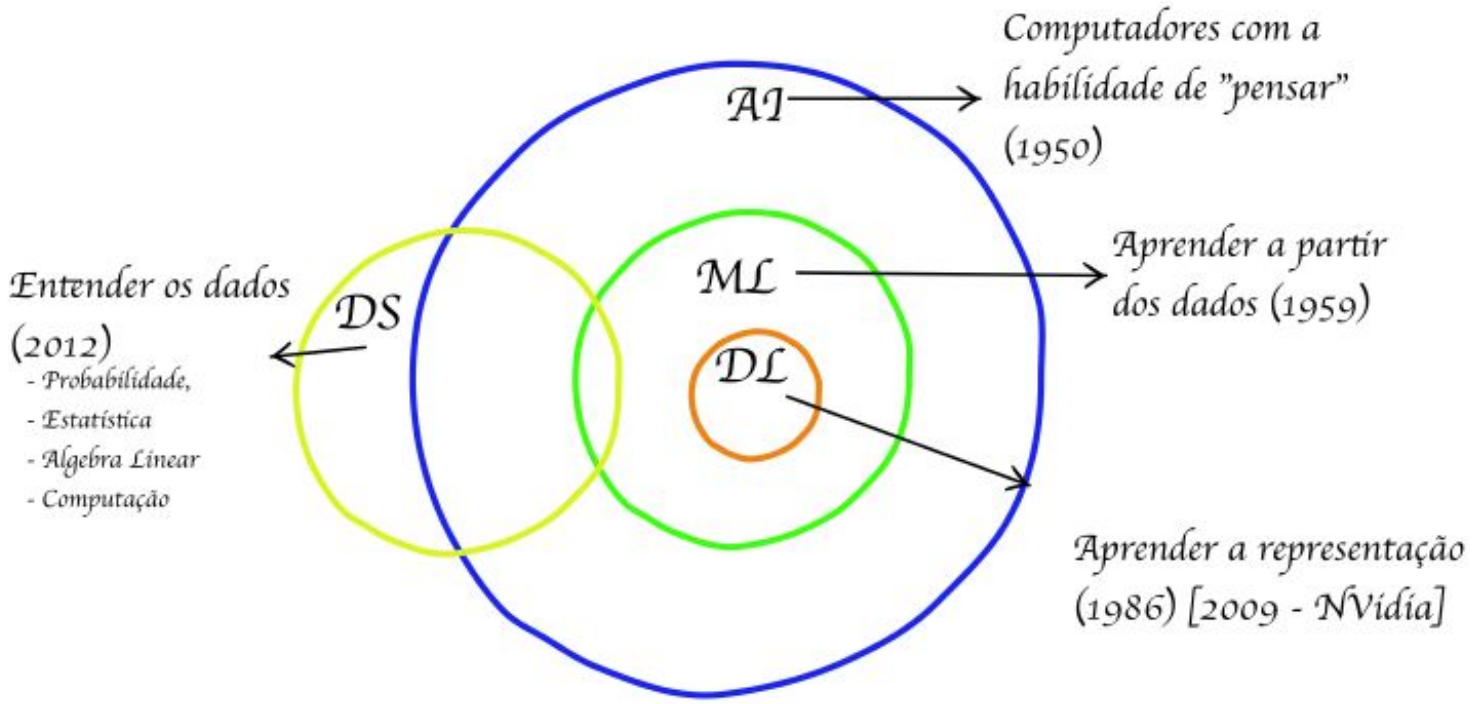
Deep Learning

PROF. ALCEU BRITTO

Deep Learning no contexto de IA e Aprendizagem de Máquina

- IA (Artificial Intelligence)
- ML (Machine Learning)
- DL (Deep Learning)
- DS (Data Science)



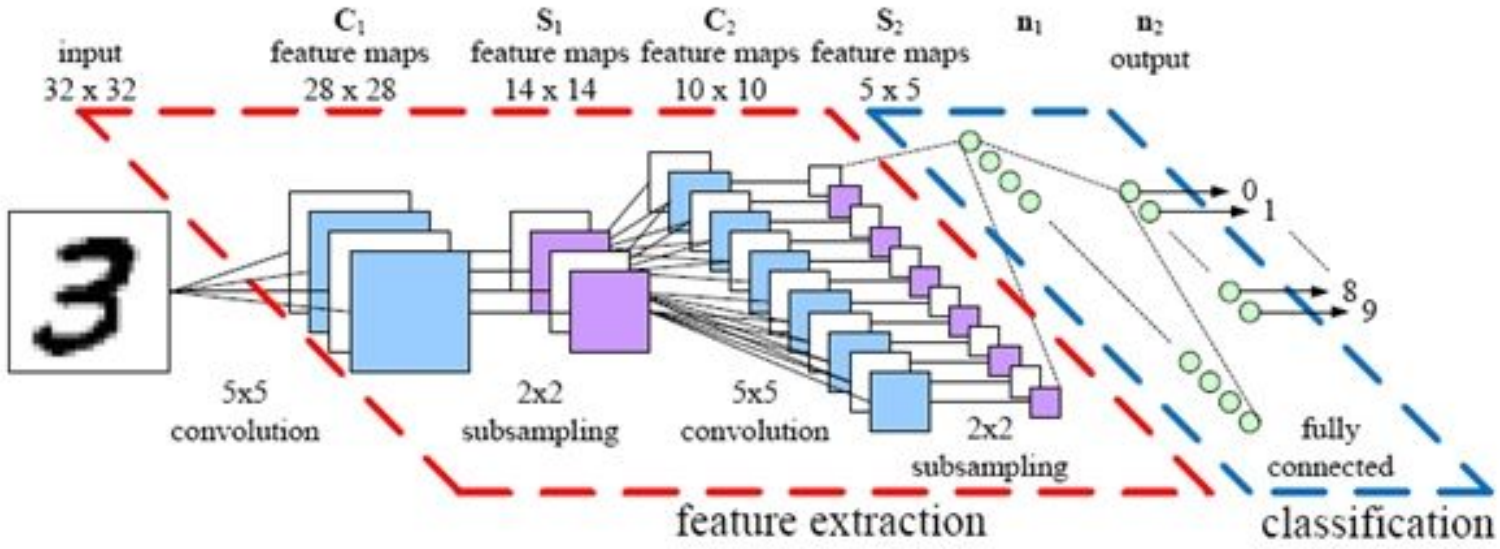
Definição CNN

- Rede Neural Convolucional (ConvNet / Convolutional Neural Network / CNN): Abordagem de aprendizado profundo proposta por (LECUN et al., 1998) e inspirada no cortex visual dos mamíferos. CNN é um modelo biologicamente inspirado pelos conceitos de campos receptivos. CNNs usam células complexas para realização de convoluções sobre o padrão de entrada e extrair características, enquanto células simples dispostas em camadas visam aprender uma tarefa de classificação ou regressão.

- Composta por camadas (layers) com funções específicas, organizadas em:
  - Camadas de convolução (Convolutional Layers)
    - Aprendem a representação do problema, a extração de características.
  - Camadas totalmente conectadas (Dense Layers)
    - Realizam a classificação (ou regressão, se for o caso)

CNN Model

Exemplo (LeNet com 7 camadas)



Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.Proceedings of the IEEE, november 1998

CNN usando Keras (Sequential)

```
# Exemplo CNN (LeNet simples)

weight_decay = 1e-4
model = Sequential()

model.add(Conv2D(4, (5,5), padding='valid', kernel_regularizer=regularizers.l2(weight_decay), activation='relu', input_shape=x_train.shape[1:]))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(12, (5,5), padding='valid', activation='relu', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(300, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

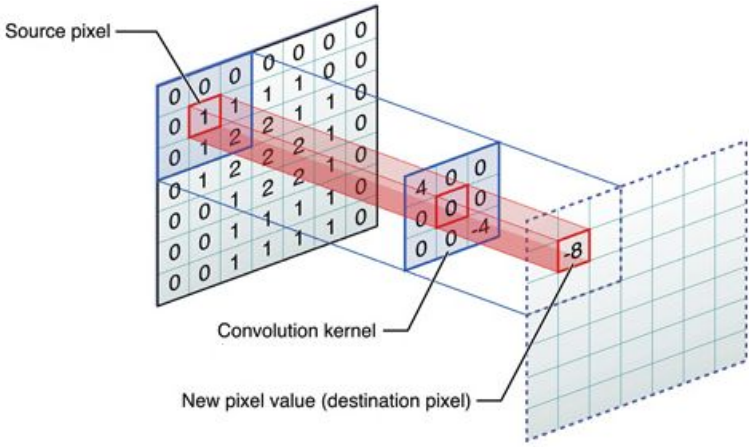
CNN usando Keras (API funcional)

```
visible = Input(shape=(64,64,1))
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(16, kernel_size=4, activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat = Flatten()(pool2)
hidden1 = Dense(10, activation='relu')(flat)
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```

Convolução

-Parâmetros principais

- Tamanho do kernel (máscara)
- Valor do passo (stride)
- Uso de padding (prenchimento)

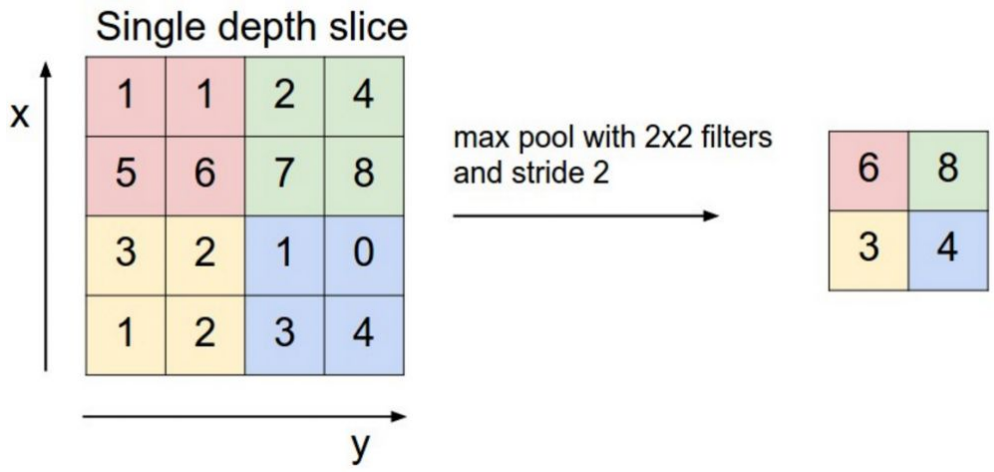


- Link para documentação do Keras

Pooling (Agregação)

Redução de escala

- Principais parâmetros
  - Tipo: Max, Avg, Mean
  - Tamanho do filtro
  - Stride



Source: cs231n.stanford.edu

- Link para documentação do Keras

Regularizadores (link para documentação do keras)

Aplicam penalidades aos parâmetros ou atividades das camadas da rede durante o treinamento.

Principais:

L1 (Lasso Regression)

Parâmetro de entrada

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

L2 (Ridge Regression)

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

Objetivo: ambos reduzem os valores dos pesos. A diferença é que o Lasso reduz para zero os coeficientes de features com pequena importância na rede já o Ridge reduz para valores pequenos porém diferentes de zero.

Regularizadores

Suponha a equação: y=Wx+b, representa uma camada (layer da rede), sendo x a entrada, W a matriz de pesos e b o bias.

- Kernel Regularizer: busca reduzir os pesos (atua em W)
- Bias Regularizer: busca reduzir o bias (atua em b)
- Activity Regularizer: busca reduzir a saída da camada (layer). Logo, reduz os pesos e ajusta o bias para obter o menor valor de Wx+b.

Batch Normalization (link para documentação do keras)

Durante o treinamento é feito o seguinte:

- 1) Cálculo de média e variância dos 'm' exemplos no batch

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{Batch variance}$$

- 2) Normalização da entrada da camada

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- 3) Ajuste (escala e shift) para obter a saída da camada

$$y_i = \gamma\overline{x_i} + \beta$$

Aprendido durante o treinamento

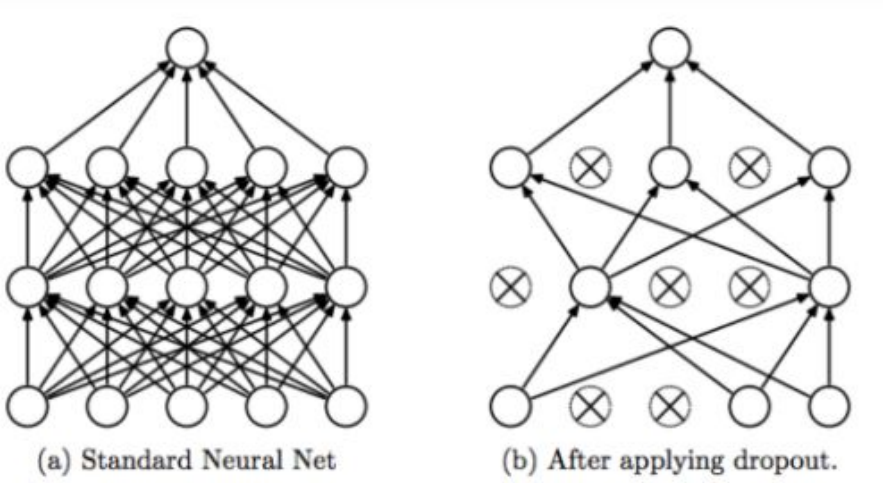
Aprendido durante o treinamento

Link para artigo original: 1502.03167.pdf (arxiv.org)

Dropout (link para documentação do keras)

Dropout proposto no artigo: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958.

- Atribui zero de forma aleatória zeros para a saída de neurônios de camadas ocultas durante o treinamento da rede.



(a) Standard Neural Net (b) After applying dropout.

Link para o artigo: srivastava14a.pdf (jmlr.org)

Compilando o modelo (Keras)

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
model.summary()
```

[Link para documentação do keras]

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_37 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
batch_normalization_21 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
dropout_4 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 14, 14, 60)        48060
_________________________________________________________________
batch_normalization_22 (Batc (None, 14, 14, 60)        240
_________________________________________________________________
dropout_5 (Dropout)          (None, 14, 14, 60)        0
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 7, 7, 60)          0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 7, 7, 128)         192128
_________________________________________________________________
batch_normalization_23 (Batc (None, 7, 7, 128)         512
_________________________________________________________________
dropout_6 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 3, 3, 128)         0
_________________________________________________________________
flatten_14 (Flatten)         (None, 1152)              0
_________________________________________________________________
dense_17 (Dense)             (None, 100)               115300
_________________________________________________________________
dropout_7 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                1010
=================================================================
Total params: 358,210
Trainable params: 357,770
Non-trainable params: 440
_________________________________________________________________
```

Treinando o modelo (Keras)

```
#model.fit para executar treinamento

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)

results = model.fit(x_train, y_train,  batch_size=batch_size, epochs=epochs,verbose=1,

        #validation_data=(x_val, y_val) validation_split=0.2)
```

- link para documentação keras (EarlyStopping)
- link para documentação keras (fit)

CNN - *Fine Tuning*

Consiste em adaptar modelo pré-treinado para um novo domínio.

- Por exemplo: adaptar CNN (VGG 16) já treinada na base Imagenet (1000 classes) para um problema de classificação de imagens contendo apenas 10 classes.

- Objetivo: Transferência do aprendizado. Treina-se em base maior e utilizada-se em domínio no qual a base é pequena.

- Estratégia comum: congela-se a parte convolucional (*Convolutional Layers, CL*), retreina-se apenas a parte *Fully Connected (FC)*

- Roteiro (*Fine Tuning –> FC Layers*)
  - 1) Escolha um modelo pré-treinado: LeNet, VGG16, VGG19, Inception, ... Há vários no Keras.

  - 2) Carregue o modelo pré-treinado selecionado, apenas a parte convolucional
    - from keras.applications import VGG16
    - vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    - vgg_conv.summary()

CNN - *Fine Tuning*

- Roteiro (*Fine Tuning –> FC Layers*)
  - 3) Adapte o modelo adicionando uma nova parte FC, considerando o número de classes do novo domínio.
    - model = Sequential()
    - model.add(vgg_conv)
    - model.add(Flatten())
    - model.add(Dense(1024, activation='relu'))
    - model.add(Dropout(0.5))
    - model.add(Dense(10, activation='softmax'))
  - 4) Congele as camadas convolucionais
    - for layer in model.layers[:-4]:
      - layer.trainable = False

CNN - Fine Tuning

- Roteiro (*Fine Tuning –> FC Layers*)
  - 5) Verifique o status das camadas
    - for layer in model.layers:
    - print(layer, layer.trainable)
  - 6) Compile o modelo adaptado
    - model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
    - model.summary()
  - 7) Retreine o modelo
    - es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
    - history = model.fit(trainX, trainY,
    - batch_size=128,
    - epochs=50,
    - verbose=1,
    - validation_split=0.2)

CNN - Fine Tuning

- Roteiro (Fine-Tunning – FC Layers)

  8) Avalie o modelo criado

  - _, acc = model.evaluate(testX, testY, verbose=0)
  - print('Final Accuracy: > %.3f' % (acc * 100.0))

Veja script de exemplo: VGG16_Fine_Tuning

Considerações finais:

- permite ajuste fino de modelo pré-treinado para novos problemas (*transfer learning*). Estratégia importante para tratar problemas onde temos poucos dados para treinar um novo modelo do zero.

- problema: modelo pode perder desempenho na tarefa anterior quando é adaptado. Ver artigo de nossa autoria:

  - https://arxiv.org/abs/1905.12082

Data Augmentation

- Aumento da base de treinamento, artificialmente.
- Objetivo: aumentar a representatividade da base de treinamento.
- Estratégia: gerar novas imagens a partir da base de treinamento original aplicando transformações (rotação, translação, filtragem, flip, ...)

```
# Training the FC Layers considering data augmentation (Conv layers still frozen)
from keras.preprocessing.image import ImageDataGenerator

# initialize the generator
train_datagen = ImageDataGenerator(rotation_range=20, zoom_range=0.15, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True,
        vertical_flip=True)

# define how the batches of training samples will be generated
train_generator = train_datagen.flow(trainX, trainY, batch_size=16)

# training
history = model.fit(train_generator, epochs=epc, verbose=1)
```

[Keras ImageDataGenerator and Data Augmentation - PyImageSearch]