Universidade do Minho
Escola de Engenharia/Departamento de Informática

Visão por computador

Uma abordagem Deep Learning

Aulas práticas

IMAGEM MÉDICA

AMBIENTE DE TRABALHO

Para executarem os exercicios que irão ser propostos devem preparar o ambiente de trabalho com o seguinte sw:

python 3.x (versão 64bits)
+numpy + scipy + scikit-learn + scikit-image + h5py + matplotlib

se tiverem o anaconda instalado basta executar: conda install ....
opcional: instalar o opencv
instalar o keras: pip install keras
instalar o theano: pip install theano
instalar o tensorflow: pip install tensorflow

Para os alunos que prefiram ter uma maquina virtual com tudo já instalado coloquei uma maquina virtual (Ubuntu 16.10+opencv+keras + ...) num servidor da DI em:
https://reposlink.di.uminho.pt/uploads/d98e0b93a1b61d7dc996fe03052468fe.file.ubuntu16.10DLalunos.zip
Essa maquina tem o anaconda instalado e está configurada com ambientes.
Para trabalharem no ambiente correcto devem executar: source activate keras-test
Devem fazer o upgrade para o keras2:
pip install git+git://github.com/fchollet/keras.git --upgrade

podem fazer as instalações e upgrades tanto utilizando o conda como o pip mas sempre dentro do ambiente keras-test

REDE CNN-2D UTILIZANDO KERAS

Vamos desenvolver uma rede **CNN – Convolution Neural Network** para fazer classificação de imagens utilizando o dataset mnist:

https://s3.amazonaws.com/img-datasets/mnist.npz

AULA3 – CNN

Bibliotecas necessárias para a execução desta aula.

```
import numpy as np
#from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
K.set_image_dim_ordering('th') #pode ser 'th' ou 'tf'
import matplotlib.pyplot as plt


# fixar random seed para se puder reproduzir os resultados
seed = 9
np.random.seed(seed)
```

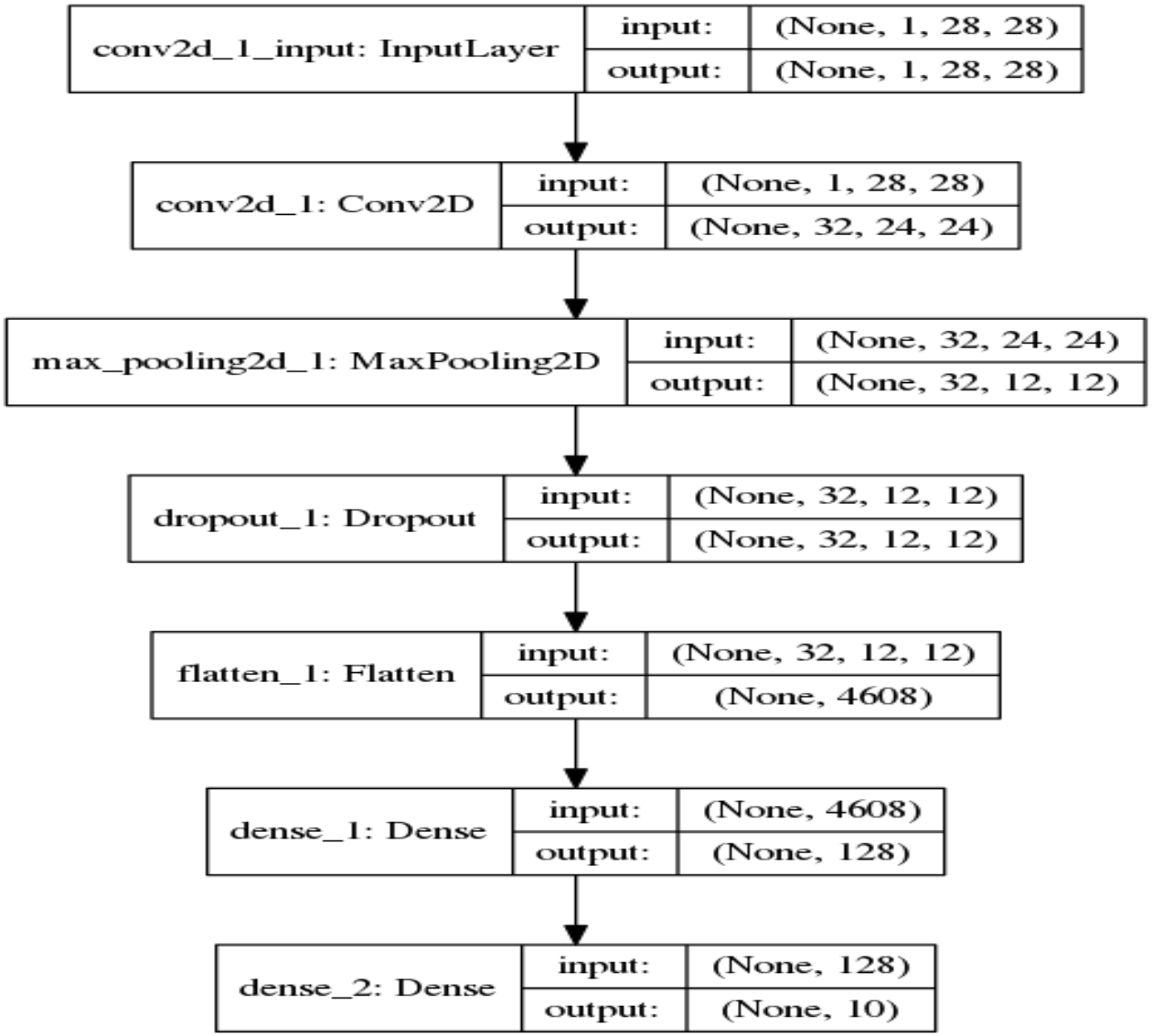
AULA3 – CNN

```
# Etapa 1 - preparar o dataset
'''
fazer o download do MNIST dataset com imagens de digitos escritos à mão para fazer a
sua classificação (já pré-preparados)
dataset: https://s3.amazonaws.com/img-datasets/mnist.npz
O ficheiro já tem tudo separado nos ficheiros {x_test.npy, x_train.npy, y_test.npy,
y_train.npy}
Os atributos de entrada estão com matrizes 3D(imagem, largura,altura) e os atributos de
saída é uma lista com o número correspondente
'''
def load_mnist_dataset(path='mnist.npz'):
    #path = get_file(path, origin='https://s3.amazonaws.com/img-datasets/mnist.npz')
    f = np.load(path)
    x_train = f['x_train']
    y_train = f['y_train']
    x_test = f['x_test']
    y_test = f['y_test']
    f.close()
    return (x_train, y_train), (x_test, y_test)
```

AULA3 – CNN

```
# Etapa 2 - Definir a topologia da rede (arquitectura do modelo) e compilar '''
criar uma rede neuronal convolucionária simples.
- a primeira camada escondida é uma camada convolucionária chamada Convolution2D.
  A camada tem 32 feature maps , cada um de dimensão 5×5 e uma função de activação 'rectifier activation function'.
  trata-se de uma camada de input, à espera de imagens com a estrutura [pixels][width][height].
- A segunda camada é de pooling que utiliza o max de MaxPooling2D.
  está configurado para uma pool size de 2×2.
- A camada seguinte é de regularização que usa Dropout.
  Está configurado para excluir aleatóriamente 20% dos neuronios na camada para reduzir overfitting.
- A camada seguinte converte os dados da matriz 2D num vector chamado Flatten.
  Assim permite-se que esse output seja tratado por uma camada completamente ligada standard.
- A camada seguinte é uma completamente ligada com 128 neuronios e uma fução de activação 'rectifier activation function'.
- Finalmente a camada de saida tem 10 neuronios correspondentes às 10 classes e uma função de activação softmax
  para apresentar na saida uma especie de probabilidade para cada classe.
- O modelo é treinado utilizando logarithmic loss e o algoritmo de gradient descent ADAM.
'''
def create_compile_model_cnn_simples(num_classes):
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

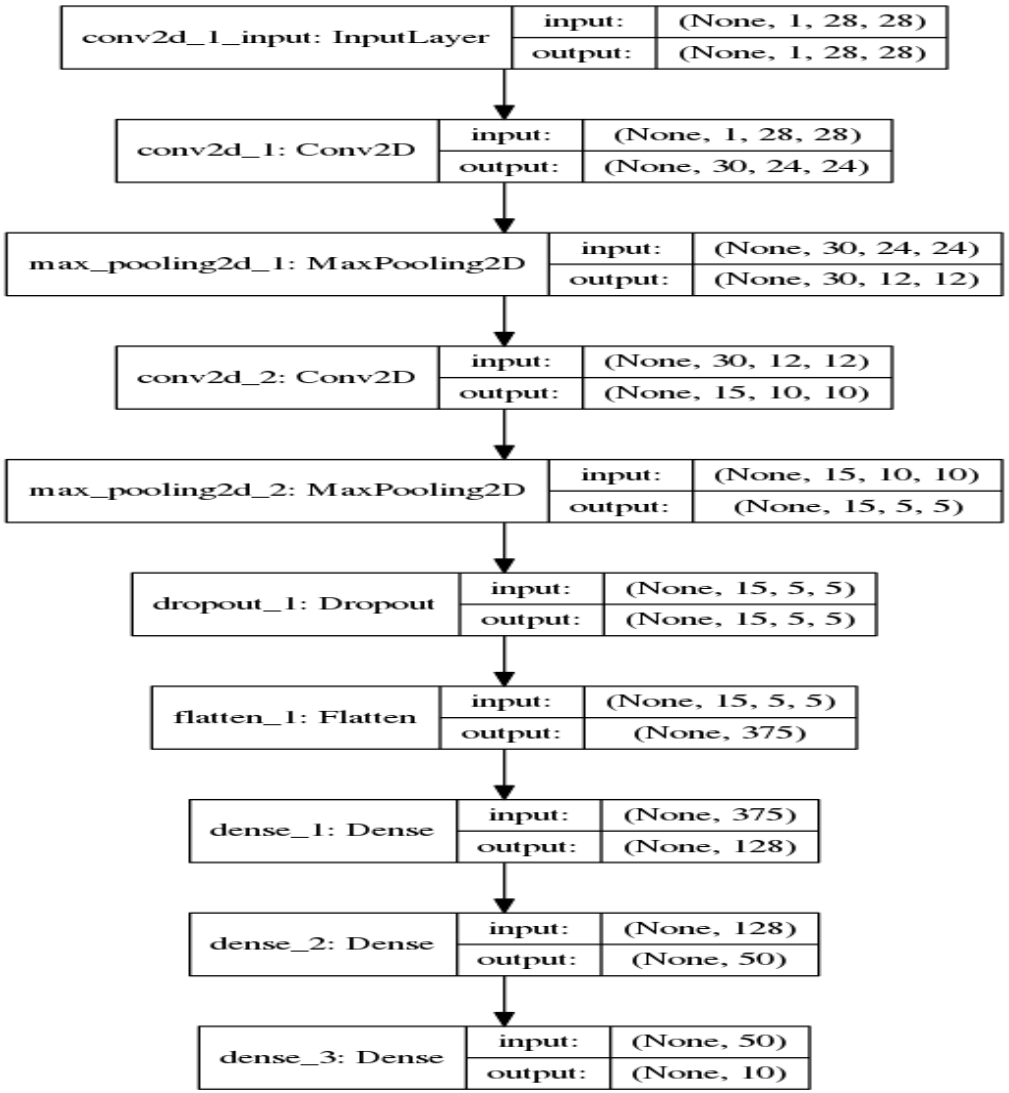
AULA3 – CNN


conv2d_1_input: InputLayer
input: (None, 1, 28, 28)
output: (None, 1, 28, 28)
conv2d_1: Conv2D
input: (None, 1, 28, 28)
output: (None, 32, 24, 24)
max_pooling2d_1: MaxPooling2D
input: (None, 32, 24, 24)
output: (None, 32, 12, 12)
dropout_1: Dropout
input: (None, 32, 12, 12)
output: (None, 32, 12, 12)
flatten_1: Flatten
input: (None, 32, 12, 12)
output: (None, 4608)
dense_1: Dense
input: (None, 4608)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 10)

AULA3 – CNN

```
# Etapa 2 - Definir a topologia da rede (arquitectura do modelo) e compilar '''
criar uma rede neuronal convolucionária mais complexa.
- Convolutional layer com 30 feature maps de dimensão 5×5.
- Pooling layer que passa o máximo de 2*2 patches.
- Convolutional layer com 15 feature maps de dimensão 3×3.
- Pooling layer que passa o máximo de 2*2 patches.
- Dropout layer com probabilidade de 20%.
- Flatten layer.
- Fully connected layer com 128 neuronios e activação rectifier.
- Fully connected layer com 50 neuronios e activação rectifier.
- Output layer.
- O modelo é treinado utilizando logarithmic loss e o algoritmo de gradient descent ADAM.
'''
def create_compile_model_cnn_plus(num_classes):
    model = Sequential()
    model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

AULA3 – CNN


conv2d_1_input: InputLayer
input: (None, 1, 28, 28)
output: (None, 1, 28, 28)
conv2d_1: Conv2D
input: (None, 1, 28, 28)
output: (None, 30, 24, 24)
max_pooling2d_1: MaxPooling2D
input: (None, 30, 24, 24)
output: (None, 30, 12, 12)
conv2d_2: Conv2D
input: (None, 30, 12, 12)
output: (None, 15, 10, 10)
max_pooling2d_2: MaxPooling2D
input: (None, 15, 10, 10)
output: (None, 15, 5, 5)
dropout_1: Dropout
input: (None, 15, 5, 5)
output: (None, 15, 5, 5)
flatten_1: Flatten
input: (None, 15, 5, 5)
output: (None, 375)
dense_1: Dense
input: (None, 375)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 50)
dense_3: Dense
input: (None, 50)
output: (None, 10)

AULA3 – CNN

```
def mnist_utilizando_cnn_simples():
    (X_train, y_train), (X_test, y_test) = load_mnist_dataset('mnist.npz')
    # transformar para o formato [instancias][pixeis][largura][altura]
    X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
    X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
    # normalizar os valores dos pixeis de 0-255 para 0-1
    X_train = X_train / 255
    X_test = X_test / 255
    # transformar o label que é um inteiro em categorias binárias, o valor passa a ser o
correspondente à posição
    # o 5 passa a ser a lista [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
    y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]
    # definir a topologia da rede e compilar
    model = create_compile_model_cnn_simples(num_classes)
    print_model(model,"model_simples.png")
    # treinar a rede
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,
batch_size=200, verbose=2)
    print_history_accuracy(history)
    #print_history_loss(history)
    # Avaliação final com os casos de teste
    scores = model.evaluate(X_test, y_test, verbose=0)
    print('Scores: ', scores)
    print("Erro modelo MLP: %.2f%%" % (100-scores[1]*100))
```

AULA3 – CNN

```
def mnist_utilizando_cnn_plus():
    (X_train, y_train), (X_test, y_test) = load_mnist_dataset('mnist.npz')
    # transformar para o formato [instancias][pixeis][largura][altura]
    X_train = X_train.reshape(X_train.shape[0], 1, 28, 28).astype('float32')
    X_test = X_test.reshape(X_test.shape[0], 1, 28, 28).astype('float32')
    # normalizar os valores dos pixeis de 0-255 para 0-1
    X_train = X_train / 255
    X_test = X_test / 255
    # transformar o label que é um inteiro em categorias binárias, o valor passa a ser o
correspondente à posição
    # o 5 passa a ser a lista [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
    y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]
    # definir a topologia da rede e compilar
    model = create_compile_model_cnn_plus(num_classes)
    print_model(model,"model_plus.png")
    # treinar a rede
    history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200,
verbose=2)
    print_history_accuracy(history)
    #print_history_loss(history)
    # Avaliação final com os casos de teste
    scores = model.evaluate(X_test, y_test, verbose=0)
    print('Scores: ', scores)
    print("Erro modelo MLP: %.2f%%" % (100-scores[1]*100))
```

AULA3 – CNN

```
if __name__ == '__main__':
    #mnist_utilizando_cnn_simples()
    mnist_utilizando_cnn_plus()
```