Machine Learning - Homework 2

RoboCup 8_Classification problem

Gianluca Pulicati - 1708686

Introduction

The second Machine Learning course's homework has been assigned to the students of Engineering Computer Science at "Sapienza Univerità di Roma" and marks the first approach to the use of Neural Networks in a very pratical problem.
The homework is based on the seminar of *"Deep Ensembles & IROS"* hold during lectures. In the seminar was discussed how Robot security is becoming important nowadays and how Deep Ensembles methodologies are useful to help ensuring this purpose. In fact, security starts from being able create a model in order to classify new instances of robot System logs as normal or abnormal thus to identify a "good" behavior from a "cyber-attacked" one. Basically with Ensemble learning is possible to train multiple models to solve the problem and then combine all the different outputs to obtain the best result.

Objectives

The main object of the homework is to classify images which belong to eight different classes:

- Bouillin cup
- Flavored Water
- Latex Gloves
- Marshmallow bag
- Nectarines
- Pickled vegetables
- Plastic food containers

- Plastic forks

To solve this task is possible to use several different methods, although one studied during lectures is probably the most suitable: Convolutional neural network. This type of network were inspired by the connectivity pattern between neurons which resembles the organization of animal visual cortex, thus are used the most to classify images, videos, natural language processing, etc.

Dataset

The dataset given for this homework consists in eight different folders, each containing a variable number of images of the relative class.
Datasets in NN are fundamental: the quantity and the quality of elements are the kernel of the NN training.
The given dataset is not a huge one and it also contains some noise elements which could be useful for the model, cause it has to learn how to deal with them.

Deep learning

The key of this homework is the use of "Deep Learning", which is an important subset of Machine Learning.
In deep learning the data is sent through different levels of neural networks, and each network hierarchically determines the specific features of the images. In other words, after processing the data, the systems finds appropriate identifiers to classify every image according to his class. A huge difference between a Machine Learning approach is that, in DL, data doesn't necessarily requires a structured/well tagged elements to use as "root" of learning process; Deep learning processes doesn't even need a human intervention meanwhile the data is crossing the layers and, at the end, the Neural network is even able to learn from his self; in fact, as a multilevel layers "algorithm", data is placed in a hierarchy of different concepts, which ultimately are able to learn from own mistakes. Meanwhile a Machine Learning algorithm would need some sort of "teaching" operations.

Tools and methodologies

To accomplish this homework these are the tools used:

```
import numpy as np
import keras
import os
from PIL import Image
from keras.preprocessing.image import ImageDataGenerator
#import tensorflow as tf

#print("Tensorflow version %s" %tf.__version__)
print("Keras version %s" %keras.__version__)

#device_name = tf.test.gpu_device_name()
#if device_name != '/device:GPU:0':
#  raise SystemError('GPU device not found')
#print('Found GPU at: {}'.format(device_name))
```

The Python's code above shows all the libraries used for the homework. As shown, *"keras"* library is imported to solve the problem, which is a open-source library that provides an interface for artificial neural networks.
Then:
PIL and os: used to import and work with images
ImageDataGenerator: function used to handle images in keras and run some pre-processing operations.

Note: The whole homework has been developed on "Google Colab", firstly to have a smooth "hardware" setup in order to test, run and debug the problem (which could be very slow on a dated computer) and also to add some layout while coding the solution. To note is the use of the "GPU" runtime provided which is an important speed-up for the training process.

Data-set file management

Select a folder, resize all images and save.

(Manually adjust settings if needed)

```
def resize_image(src_img, size=(224,224), bg_color="white"):

    # rescale the image so the longest edge is the right size
    src_img.thumbnail(size, Image.ANTIALIAS)

    # Create a new image of the right shape
    new_image = Image.new("RGB", size, bg_color)

    # Paste the rescaled image onto the new centered background
    new_image.paste(src_img, (int((size[0] - src_img.size[0]) / 2), int((size[1] - src_img.size[1]) / 2)))

    # return the resized image
    return new_image


# get the list of test image files
test_folder = '/content/drive/MyDrive/8_class_classification/train/plastic_for'
test_image_files = os.listdir(test_folder)

# Initialize index for name files
index = 0

# Empty array on which to store the images
image_arrays = []
size = (224,224)
background_color="white"

# Get the images
for file_idx in range(len(test_image_files)):
    img = Image.open(os.path.join(test_folder, test_image_files[file_idx]))

    # resize the image
    resized_img = np.array(resize_image(img, size, background_color))

    # Add the image to the array of images
    image_arrays.append(resized_img)
```

In this piece of code is shown how dataset has been pre-processed.
First of all there was the need of handling all different size of images given; basically with this function the program runs a "resize" operation on every image of the selected folder and save the new image in a new directory. To do this it create a new "white-empty background" of *(224,224,3)* size to start, and then tries to adapt the original image to it.
This operation is repeated for every class folder given.
At the end the files are also manually splitted in two subsets: one containing the training set (70%) and the other containing the validation set (30%); these sets are kept fixed in order to have more comparable metrics of models.

Data generation and augmentation

```
from google.colab import drive
drive.mount('/content/drive',)

datadir = '/content/drive/MyDrive/Resized_images'
trainingset = datadir+'/train/'
testset = datadir + '/test/'

batch_size = 32
train_datagen = ImageDataGenerator(
    rescale = 1. / 255,\
    zoom_range=0.1,\
    rotation_range=10,\
    width_shift_range=0.1,\
    height_shift_range=0.1,\
    horizontal_flip=True,\
    vertical_flip=False)

train_generator = train_datagen.flow_from_directory(
    directory=trainingset,
    target_size=(224, 224),
    color_mode="rgb",
    batch_size=batch_size,
    class_mode="categorical",
    shuffle=True,
    seed=42)
```

Next up there is the loading of dataset into keras. Here batch size value is selected and then augmentation of files start: data is replaced through "flow from directory" and "image data generator" functions.
Data augmentation is another fundamental step in Deep Learning; it's used to enlarge the dataset with different shape of same files. For example images are rotated, flipped and zoomed thus the Neural network will have more values to use as training elements. This is done also for test dataset. (not shown above).

AlexNet model

Define model Alexnet

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten,\
                         Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras import regularizers
from keras import optimizers

def AlexNet(input_shape, num_classes, regl2 = 0.0001, lr=0.0002):

    model = Sequential()

    # C1 Convolutional Layer
    model.add(Conv2D(filters=96, input_shape=input_shape, kernel_size=(11,11),\
                     strides=(4,4), padding='same'))
    # Batch Normalisation before passing it to the next layer
    model.add(BatchNormalization())

    model.add(Activation('relu'))
    # Pooling
    #####model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='valid'))
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

    # C2 Convolutional Layer
    model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same'))

    # Batch Normalisation
    model.add(BatchNormalization())

    model.add(Activation('relu'))
    # Pooling
    model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

    # C3 Convolutional Layer
    model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))

    # Batch Normalisation
    model.add(BatchNormalization())

    model.add(Activation('relu'))
```

```
# C4 Convolutional Layer
model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))

# Batch Normalisation
model.add(BatchNormalization())

model.add(Activation('relu'))

# C5 Convolutional Layer
model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'))

# Batch Normalisation
model.add(BatchNormalization())

model.add(Activation('relu'))
# Pooling
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

# Flatten
model.add(Flatten())

flatten_shape = (input_shape[0]*input_shape[1]*input_shape[2],)
print(flatten_shape)

# D1 Dense Layer
model.add(Dense(4096, input_shape=flatten_shape, kernel_regularizer=regularizers.l2(regl2)))

# Batch Normalisation
model.add(BatchNormalization())

model.add(Activation('relu'))
# Dropout
model.add(Dropout(0.4))

# D2 Dense Layer
model.add(Dense(4096, kernel_regularizer=regularizers.l2(regl2)))

# Batch Normalisation
model.add(BatchNormalization())

model.add(Activation('relu'))
# Dropout
model.add(Dropout(0.3))

# D3 Dense Layer
```

```
    model.add(Dense(1000,kernel_regularizer=regularizers.l2(regl2)))

    # Batch Normalisation
    model.add(BatchNormalization())

    model.add(Activation('relu'))
    # Dropout
    model.add(Dropout(0.2))

    # Output Layer
    model.add(Dense(num_classes))
    # Batch Normalisation
    model.add(BatchNormalization())

    model.add(Activation('softmax'))

    # Compile

    adam = optimizers.Adam(lr=lr)
    model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

    return model

# create the model
model = AlexNet(input_shape,num_classes)
model.summary()
```

LenNet model

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten,\
                         Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers.normalization import BatchNormalization
from keras import regularizers
from keras import optimizers

def LeNet(input_shape, num_classes):

    print('\nLeNet model')
    model = Sequential()

    print('\tC1: Convolutional 6 kernels 5x5')
    model.add(Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activation='tanh', input_shape=input_shape, padding='same'))
    print('\tS2: Average Pooling 2x2 stride 2x2')
    model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
    print('\tC3: Convolutional 16 kernels 5x5')
    model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
    print('\tS4: Average Pooling 2x2 stride 2x2')
    model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
    print('\tC5: Convolutional 120 kernels 5x5')
    model.add(Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
    model.add(Flatten())
    print('\tF6: Fully connected, 84 units')
    model.add(Dense(84, activation='tanh'))
    print('\tF7: Fully connected, 10 units')
    model.add(Dense(num_classes, activation='softmax'))

    optimizer = 'adam' #alternative 'SGD'
    model.compile(loss=keras.losses.categorical_crossentropy, optimizer=optimizer, metrics=['accuracy'])

    return model

# create the model
model = LeNet(input_shape,num_classes)
model.summary()
```

Conclusions

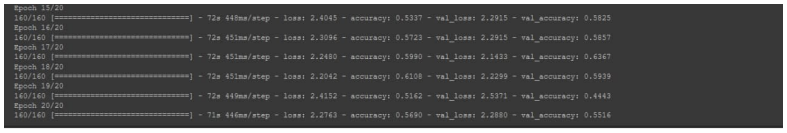
Working with Neural network brings, in general, a set of choice that needs to be taken. As described above, in Deep Learning one of the hardest part is to work with the data-set: building a balanced and large one is the key to have the best accuracy on the trained model.
In this homework, thought, the data-set is almost fixed and pre-process' operations brings relative small changes. Since the solution develops around Neural network it's also good to not completely remove noise data; in "real life" problems every dataset has some noise, thus the model had to deal and work around it. For this reason the only pre-process' operation done in this homework was about adjusting the size of every image to a standard one, this is also to reduce the total computational time needed by the program to train the model.

The first model trained to solve the problem is "LeNet": it is composed by some Convolutional layers alternated with Pooling ones, ending with two Dense layers. It was trained mostly with standard Batch_size (32) value and both with a relative low epoch value (10) and high one (30).
Unfortunately the accuracy level reached by the model was not very good but i expected that: LeNet was not designed to work with images of relative high resolution, in first place it was used to classify some "digits" images of about (28x28x1) size, the data-set of this homework has higher complexity which cannot be evaluated by the settings used in LeNet layers. Even trying to playing with some hyper-parameters didn't have a big impact on final metrics.

For this reasons another model was tested: "AlexNet". At first glance was possible to check that AlexNet general settings are much more aligned to the problem we are trying to resolve. AlexNet is a CNN which uses ReLU as activation function, it takes (224x224x3) (RGB images!) as imput for first layer. Also, at the end, AlexNet has two layers fully connected with 4096 neurons each which are important cause they apply Dropout normalization. Basically with the Dropout we also let the training process "learn from noisy", when dropped out some layers output are simply ignored. This has the effect of making the layer look-like a layer with a different number of nodes to the prior layer. This simulates like a random sparse activation from a given layer and turns the system to learn from this "noise" operation!
Although in this case especially playing with Dropout value had both a good and bad response. By turning the value down the model acquired slightly better accuracy point, thus brought me to the idea that the AlexNet is having some problem to learn from this noise data generated by layers, probably cause the data-

self itself has a large value of "dirty" images which created some misclassification. In general, Alexnet code provided during lectures was used for this homework, but i tried adjusting some value (especially "padding" and "strides" ones) after studying the composition of the layers of the model found on web. Also trying to edit the learning rate of loss optimizer Adam wasn't successful, slightly better results were obtained by augmenting it a little bit. (0.0002)
At the end is possible to say that even if the total accuracy (val_accuracy) reached is not "very high", the solution provided with AlexNet at least reached a very balanced result, showing also that "plastic_food_container" was class creating a lot of problems and probably needed some more pre-process' operations.

AlexNet




```
Found 2199 images belonging to 8 classes.
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/training.py:1905:
  warnings.warn('`Model.predict_generator` is deprecated and '
69/69 [==========] - 8s 84ms/step
True                Predicted          errors  err %
------------------------------------------------------------------
pickled_vegetables ->  plastic_food_container   339     15.42 %
Flavored_Water   ->  plastic_food_container     319     14.51 %
Latex_Gloves     ->  plastic_food_container     314     14.28 %
Nectarines       ->  plastic_food_container     311     14.14 %
Marshmallows_bag ->  plastic_food_container     272     12.37 %
plastic_fork     ->  plastic_food_container     214     9.73 %
bouillin_cup     ->  plastic_food_container     99      4.50 %
plastic_food_container ->  Latex_Gloves         16      0.73 %
Nectarines       ->  Latex_Gloves       11      0.50 %
pickled_vegetables ->  Latex_Gloves             8       0.36 %
plastic_fork     ->  Latex_Gloves       8       0.36 %
plastic_fork     ->  Flavored_Water     4       0.18 %
Flavored_Water   ->  Latex_Gloves       4       0.18 %
Nectarines       ->  Flavored_Water     4       0.18 %
pickled_vegetables ->  Flavored_Water           3       0.14 %
Marshmallows_bag ->  Latex_Gloves       3       0.14 %
plastic_food_container ->  Flavored_Water       3       0.14 %
Latex_Gloves     ->  Flavored_Water     3       0.14 %
Marshmallows_bag ->  Flavored_Water     2       0.09 %
bouillin_cup     ->  Latex_Gloves       2       0.09 %
bouillin_cup     ->  Flavored_Water     2       0.09 %
```