Image Classification regarding objects in a home environment for the RoboCup@Home competition

Matteo Facci
1597454

October 6, 2022

1 Introduction

This report provides some solutions for the classification problem of images regarding objects in a home environment.
To this end, the RoboCup@Home-Objects dataset is used, which has been developed within the RoboCup@Home competition. Starting from the manipulation of the latter, in order to prepare the environment where the program will be executed, new methods for training a classification model and predicting new results will be shown and compared.
There is a total amount of 8 classes :

```
classes = ['glass food container', 'potato crisps', 'sports drinking water','pasta sides','replacement heads',
'steak knives', 'plums', 'side dish plate'];
```

2 Programming Language

The program is developed in python in order to make use of *Keras* Library, a powerful library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries *Theano* and *TensorFlow* and allows to define and train neural network models in just a few lines of code.
Colab is used as a development environment. It allows to write and execute arbitrary python code through the browser, and it is especially well suited to machine learning and data analysis. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.
Moreover a particular section of the report has been developed with the help of Matlab and its Deep Learning Toolbox, which provides a framework for designing and implementing deep neural networks with algorithms, pretrained models, and useful apps.

3 Data pre-processing and data augmentation

The Dataset consists on 8779 images divided not equally for each of the 8 categories mentioned above.
That is very few examples to learn from for a classification problem, due to the disproportionate number of classes compared to the insufficient number of images. So this is a

challenging machine learning problem, but it is also a realistic one: in a lot of real-world use cases, data collection can be extremely expensive or sometimes near-impossible.
It can almost be considered an unrepresentative dataset. An unrepresentative dataset means a dataset that may not capture the statistical characteristics relative to another dataset drawn from the same domain, such as between a train and a validation dataset. This can commonly occur if the number of samples in a dataset is too small, relative to another dataset.
In order to make the most of the few training examples, they are "augmented" via a number of random transformations, so that the model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.

3.1 Data augmentation in Python

In *Keras* this can be done via the *ImageDataGenerator* class. This class allows to:

- configure random transformations and normalization operations to be done on your image data during training
- instantiate generators of augmented image batches (and their labels) via `flow_from_directory(directory)`.
  These generators can then be used with the *Keras* model methods that accept data generators as inputs, `fit_generator`, `evaluate_generator` and `predict_generator`.

For this purpose the contents of the dataset folder have been divided into two subfolders, the *train* folder, containing the 80% of the total amount of the images in the dataset (7026 training samples from 8 classes), and the *test* folder, containing the remaining 20% (1753 test samples from 8 classes).
The final structure of the folder is illustrated below.

```
train
├── plums
├── glass food container
├── potato crisps
├── sports drinking water
├── pasta sides
├── replacement heads
├── steak knives
├── plums
└── side dish plate
test
├── plums
├── ...
└── side dish plate
```

All the images are loaded in color mode RGB with batch size 32. The target size of *ImageDataGenerator* is one of the parameters which has been changed in the experiments basing on the input shape of the Convolutional Neural Network in exam. When this parameter is present, the images are stretched to the target size.

3.2 Data augmentation in Matlab

The logic in Matlab is practically the same but, thanks to the Deep Learning Toolbox, each operation can be performed with a single function.
`augmentedImageDatastore` transforms batches of training and test data, with optional preprocessing such as resizing, rotation, and reflection. It resizes images to make them compatible with the input size of the network in exam.
In Matlab, the contents of the dataset folder have not been divided into subfolders for *train* and *test*, since the function `splitEachLabel` operates the division automatically. Also in this case, the 80% of the total amount of the images are used for training and the remaining 20% for testing. All the images are loaded in color mode RGB with batch size 32.

4 Training a neural network from scratch

First a new neural network from scratch is built for doing classification on the dataset. This will give an initial baseline to compare to other techniques used and to other models of neural networks. For sake of simplicity it has been called $Netfs$ (from $Net\,from\,scratch$).
One of the simplest neural networks is $LeNet$, but certainly, given its structure, this does not allow to classify in a sufficiently accurate way the type of images contained in the dataset. However, this can be a starting point for the creation of $Netfs$.
This is the difference between the $LeNet$ layers and the new $Netfs$' ones:

LeNet	Netfs
Convolutional 6 kernels 5x5	Convolutional 12 kernels 11x11
Average Pooling 2x2 stride 2x2	Average Pooling 2x2 stride 2x2
Convolutional 16 kernels 5x5	Convolutional 32 kernels 8x8
Average Pooling 2x2 stride 2x2	Average Pooling 2x2 stride 2x2
Convolutional 120 kernels 5x5	Convolutional 240 kernels 8x8
Fully connected, 84 units	Fully connected, 2048 units (dropout 0.5)
-	Fully connected, 512 units
Fully connected, 24 units	Fully connected, 8 units

All of the layers in $Netfs$ use $ReLU$ as activation function except for the last layer which uses $softmax$, as in $LeNet$. The size of kernels has been enlarged in order to test if this can be a tuning that allows the net to perform better. After all layers, $BatchNormalization$ and a larger *dense* layer have been added, and this helped to create a deeper network which performed better.
In fact a deeper neural network needs less neurons to reach sufficient results, so it is a way to increase the results quality without increasing the training effort and the convergence time.

The loss function will be $categorical\,cross-entropy\,loss$, and the learning algorithm will be $Adam$. Various things about this network can be changed to get better performance, perhaps using a larger network or a different optimizer will help, but for the purposes, the goal is to get an approximate baseline for comparison's sake.
During the training process, after the creation of the model, in order to prevent overfitting, `ModelCheckpoint` and `EarlyStopping` methods are imported from $Keras$. An object of both is created and passed as callback function to `fit_generator`. `ModelCheckpoint` allows to save the model by monitoring a specific parameter (validation accuracy in this case).
The model will only be saved if the validation accuracy in the current epoch is greater than the one in the last epoch. `EarlyStopping` instead stops the training of the model earlier if there is no increase in the parameter monitored. The *patience* parameter is set to 15 which means that the model will stop to train if it does not see any rise in validation accuracy in 15 epochs.
The same training approach has been used also for training from scratch two state-of-the-art neural networks for comparison's purpose.

4.1 Netfs

- $input\,shape$: 224×224
- $total\,params$: 2078872
- $trainable\,params$: 2073184
- $non-trainable\,params$: 5688
- $epochs$: 80

Metric	Value
$Accuracy$	0.651
$Precision$	0.678
$Recall$	0.653
$F1\,Score$	0.658

4.2 AlexNet

- $input\,shape$: 227×227
- $total\,params$: 35427792
- $trainable\,params$: 35406656
- $non-trainable\,params$: 21136
- $epochs$: 40

Metric	Value
$Accuracy$	0.610
$Precision$	0.637
$Recall$	0.610
$F1\,Score$	0.614

4.3 VGG16

- $input\,shape$: 224×224
- $total\,params$: 134293320
- $trainable\,params$: 134293320
- $non-trainable\,params$: 0
- $epochs$: 40

Metric	Value
$Accuracy$	0.681
$Precision$	0.693
$Recall$	0.681
$F1\,Score$	0.684

5 Transfer Learning

Another approach to the image classification problem is to fine-tune deeper layers by training the network on the new dataset with the pre-trained network as a starting point.
Fine-tuning a network with transfer learning is often faster and easier than constructing and training a new network from scratch. The network has already learned a rich set of image features, but in this way, it can learn features specific to the new dataset.
This method usually gives the highest accuracy and using a pre-trained network with transfer learning is typically much faster and easier than training a network from scratch.
In the present case a $VGG16$ model pre-trained with ImageNet has been used in order to transfer learning to a new neural network called $TransferNet$.
The classification before the transfer occurs in the second part of the $VGG16$'s model, which takes the image features in input and picks a category.
This classifier part contains two hidden $fully\,connected$ (or $dense$) layers, each with 4096 neurons and a dense layer with 1000 neurons, one for each of the ImageNet categories.
The $softmax$ activation function is used for these neurons, so that the 1000 values they spit out sum up to unity, and can be considered as probabilities. The transfer learning procedure for VGG16 and the other pre-trained models, consists on downloading the model with the weights resulting from the training on ImageNet.
Then the classifier part is replaced by the $TransferNet$ classifier, adapted to the present problem. This classifier will have 8 output neurons in the last layer, one for each class of the dataset. Finally, all the layers of the convolutional part will be freezed, so that only the parameters of the new classifier will be trained.

5.1 ImageNet Database

The majority of the pre-trained networks are trained on a subset of the ImageNet database, which is used in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC).
It features over 14 million images sorted in 1000 object categories.

5.2 VGG16-TransferNet

- $input\,shape$: 224×224
- $total\,params$: 19855348
- $trainable\,params$: 7449492
- $non-trainable\,params$: 12405856
- $epochs$: 48

Metric	Value
$Accuracy$	0.785
$Precision$	0.804
$Recall$	0.782
$F1\,Score$	0.788

6 Feature Extraction

Feature extraction is an easy and fast way to use the power of deep learning without investing time and effort into training a full network.
It only requires a single pass over the training images, therefore it is especially useful in absence of a GPU.
In this regard, the entire feature extraction operation was carried out in this case using Matlab for comparison, on a laptop with single CPU (Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz - RAM 8GB).
The learned image features are extracted by using a pre-trained network, and then those features are used to train a classifier, such as a support vector machine (SVM) using `fitcecoc`, a useful function for fitting multiclass models included in the Statistics and Machine Learning Toolbox.
Pros of this technique are: a fast training and, if the data is very similar to the original data (ImageNet), then the more specific features extracted deeper in the network are likely to be useful for the new task. About the cons: if the data is very different from the original data (ImageNet), then the features extracted deeper in the network might be less useful for the new task.
Taking into account the following graph, all the pre-trained networks on ImageNet represented there have been used for the feature extraction, and in this case, in addition to the results also the feature extraction and training time has been reported.

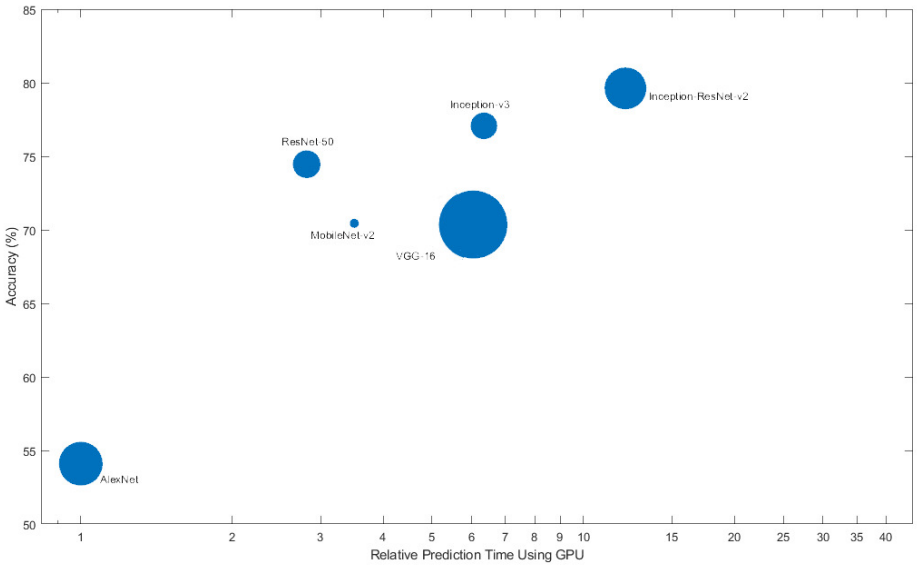

Figure 1: Indication of the relative speeds (using GPU) and accuracy of the different networks trained on the ImageNet dataset. The greater the circle, the greater the weight of the net.

6.1 Alexnet

$Training\,time = 102.547456\,s$

Metric	Value
$Precision$	0.727564
$Recall$	0.792519
$F1\,Score$	0.724455

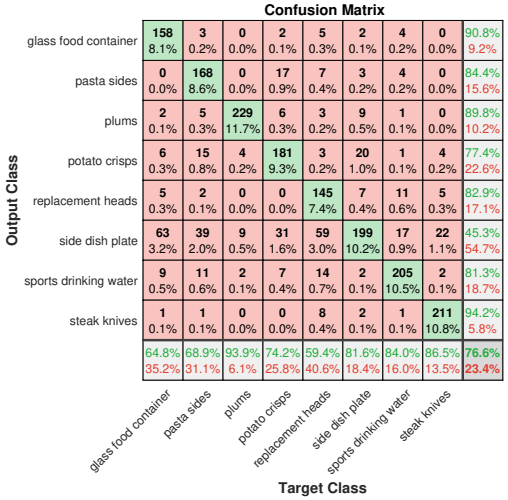

Figure 2: $Confusion\,Matrix$ for SVM using pre-trained AlexNet.

6.2 VGG16

$Training\,time = 1800.486780\,s$

Metric	Value
$Precision$	0.805769
$Recall$	0.811865
$F1\,Score$	0.806679

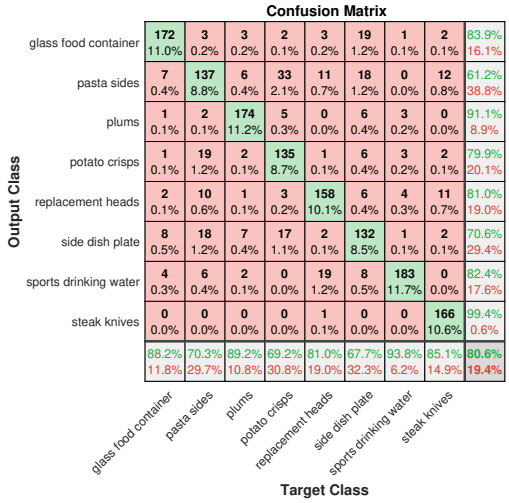

Figure 3: $Confusion\,Matrix$ for SVM using pre-trained VGG16.

6.3 MobileNet-v2

$Training\,time = 331.062729\,s$

Metric	Value
$Precision$	0.823770
$Recall$	0.826767
$F1\,Score$	0.823620

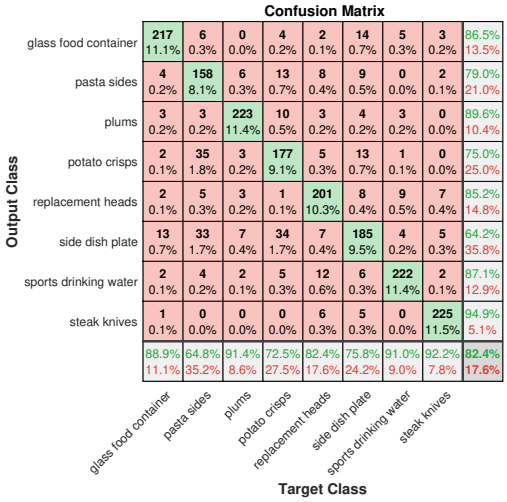

Figure 4: *Confusion Matrix* for SVM using pre-trained MobileNet-v2.

6.4 ResNet-50

$Training\,time = 633.149611\,s$

Metric	Value
Precision	0.866667
Recall	0.875311
F1 Score	0.868155

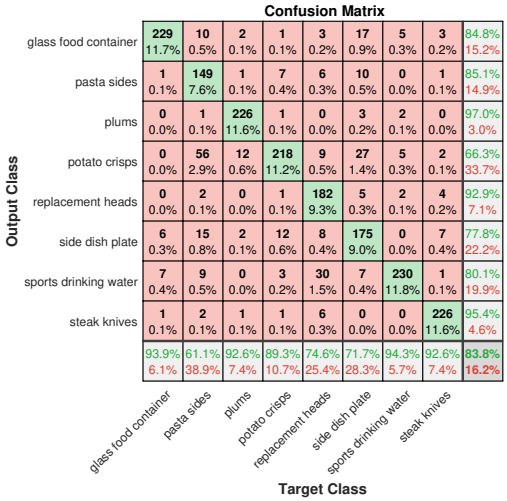

Figure 5: *Confusion Matrix* for SVM using pre-trained ResNet-50.

6.5 Inception-v3

$Training\,time = 1256.542420\,s$

Metric	Value
$Precision$	0.836066
$Recall$	0.838652
$F1\,Score$	0.835127

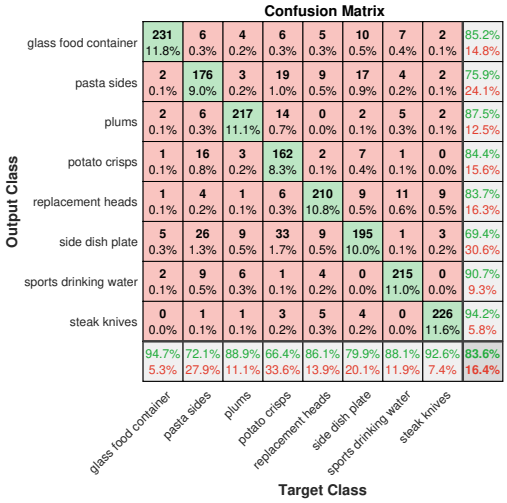

Figure 6: $Confusion\,Matrix$ for SVM using pre-trained Inception-v3.

6.6 Inception-Resnet-v2

$Training\,time = 2151.029954\,s$

Metric	Value
$Precision$	0.842213
$Recall$	0.853586
$F1\,Score$	0.842395

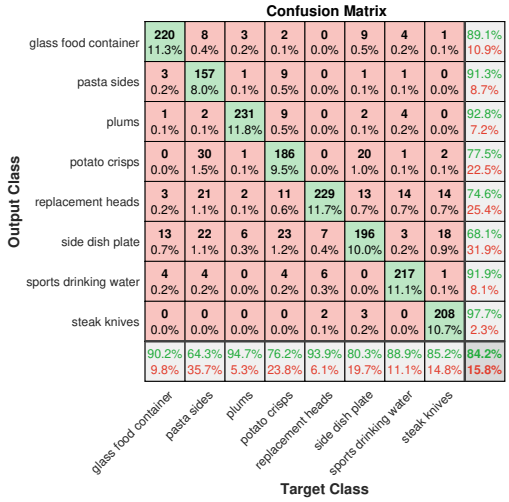

Figure 7: $Confusion\,Matrix$ for SVM using pre-trained Inception-Resnet-v2.

7 Conclusions

Collected all the data resulted from the usage of different techniques and methods for solving the present image classification problem it is possible to formulate some considerations.
Looking separately at each of the three methods used, the first analyzed is to train a neural network from scratch.
This is the method which provided the worst results among the three, in fact, due probably to the quite unrepresentative dataset and to the consequent risk of overfitting, the best result achieved was the one of the state-of-the-art $VGG16$, with a $F1\,Score$ of 0.684, slightly above the one obtained by $Netfs$. Poor $F1\,Score$ of 0.614 for $AlexNet$.
Not exactly satisfactory results accompanied by a training time exceeding 2 hours for all three models analyzed.
By using a pre-trained $VGG16$ model on ImageNet and performing the Transfer Learning technique with the $TransferNet$ neural network a 10% higher score is obtained, in particular an $F1\,Score$ of 0.788, and a certainly reduced time but always much higher than in the case of the Feature Extraction.
In this case, looking at the confusion matrices and at the $F1\,Score$ of each model, it is evident that Support Vector Machine models perform the best (for the given problem, with the given dataset) in terms of correct predictions, since their scores are of 0.868155 for the $ResNet-50$-based model, the highest score obtained among the ones of all the analyzed classifiers, and of 0.724455 for the worst case given also in this case by $AlexNet$.
As anticipated, the feature extraction and training time for the case of the best model, the $ResNet-50$-based model, is 633.149611 s, extremely reduced if compared to the times obtained with previous techniques. And considering also the limited hardware used, the technique and the model in question are among the best alternatives to solve the problem in the shortest possible time, and with the least waste of resources. Thanks also to the ease of use of Matlab and its prepared toolboxes.