



Visual classification of dumpsters with capsule networks

Francisco J. Garcia-Espinosa¹ · David Concha¹ · Juan J. Pantrigo¹ · Alfredo Cuesta-Infante¹ 

Received: 4 May 2021 / Revised: 11 February 2022 / Accepted: 9 March 2022 /
Published online: 8 April 2022
© The Author(s) 2022

Abstract

Garbage management is an essential task in the everyday life of a city. In many countries, dumpsters are owned and deployed by the public administration. An updated what-and-where list is in the core of the decision making process when it comes to remove or renew them. Moreover, it may give extra information to other analytics in a smart city context. In this paper, we present a capsule network-based architecture to automate the visual classification of dumpsters. We propose different network hyperparameter settings, such as reducing convolutional kernel size and increasing convolution layers. We also try several data augmentation strategies, as crop and flip image transformations. We succeed in reducing the number of network parameters by 85% with respect to the best previous method, thus decreasing the required training time and making the whole process suitable for low cost and embedded software architectures. In addition, the paper provides an extensive experimental analysis including an ablation study that illustrates the contribution of each component in the proposed method. Our proposal is compared with the state-of-the-art method, which is based on a Google Inception V3 architecture pretrained with Imagenet. Experimental results show that our proposal achieves a 95.35% accuracy, 2.35% over the previous best method.

Keywords Capsule networks · Image recognition · Garbage management · Smart cities

✉ Alfredo Cuesta-Infante
alfredo.cuesta@urjc.es

Francisco J. Garcia-Espinosa
franciscojose.garcia@urjc.es

David Concha
david.concha@urjc.es

Juan J. Pantrigo
juanjose.pantrigo@urjc.es

¹ Escuela Técnica Superior de Ingeniería Informática, Universidad Rey Juan Carlos, Móstoles, Spain

1 Introduction

The performance currently attained in computer vision tasks due to deep neural networks has outranked humans in many aspects; being detection, inpainting, and image generation some of the most celebrated ones. Recognition, a real challenge less than a decade ago, does not currently attract as much attention as the tasks mentioned above. Arguably, the proliferation of software libraries such as Tensorflow and PyTorch has made training deep neural networks for recognition much easier. But, how much effective are they in actual problems? Nowadays, recognition tasks are ubiquitous in each and every of the economic sectors, but most of them are *Intellectual Property* (IP) of the client that asked and paid for them. As a consequence, successes are seldom reported, depriving the community of that knowledge.

In this paper we cope with the recognition of public dumpsters and garbage containers in collaboration with Ecoembes¹, the non-profit organization that cares for the environment through recycling and eco-design of packaging in Spain. Ecoembes keeps a large database of labeled images taken in-the-wild of all the dumpsters deployed in a mid-size city for the sake of inventory and control. The effort to label such a database clearly does not scale to all the territory, with more than 11700 ZIP codes. Hence, automated recognition is a task of great interest to the organization as well as to the contractor of their services, normally the city council, but poses a number of difficulties. First, dumpsters for the same purpose show several sizes, shapes and designs. Second, they are often treated without much care by operators and suffer numerous bumps with the garbage truck and other elements in the street. Besides, in some countries, like Spain, they are 24×7 out in the street, suffering the erosion by weather. Third, sometimes dumpsters are overloaded in a way that a large surface of them are partially occluded. Other elements, different from garbage, such as motorbikes are sources of partial occlusion too. Finally, dumpsters are the frequent target of vandalism, ranging from graffiti to burning.

This problem has been previously tackled with state-of-the-art convolutional networks in [11], attaining a 93% of accuracy. On one hand, this result is quite low compared with the accuracy reported in other benchmark datasets such as MNIST, CIFAR or ImageNet. On the other hand, in many real applications, every single percent point gained can make a difference. For instance, in cybersecurity detecting 1% more intrusions, or in loan granting rejecting 1% more defaulters, can save much money and prestige loss for a company. In this case, with a data set of about 25000 dumpsters deployed throughout the city, increasing the accuracy 1% means 250 more correct classifications. Since a single dumpster gives service to a small community, the number of people benefiting from each correct labeling gained is large enough to be important for the public administration because each mistake triggers different problems such as mismanagement of material resources, improper spending and double discomfort of the neighborhood if it implies unnecessary duplication instead of a necessary substitution.

The solution introduced in this paper is able to improve the accuracy up to 95.35%, and consists of a Capsule Network architecture (Capsnet). This architecture was initially proposed in [12] and [6] as an approach to obtain high level features, different from pooling the features extracted in successive convolutions as in Google Inception [13] or ResNet [5], to mention only those that excelled at recognition tasks. To this end, Capsnets were designed with the following novel properties with respect to the rest of deep architectures: (i) the

¹www.ecoembes.com

feature maps are chunked into *primary capsules*, elements in a vector space, (ii) a number of linear transformations projects these primary capsules into a higher dimensional vector space, and (iii) a ‘routing algorithm’ clusters all the elements resulting from the previous step. The resulting features are vectors that not only capture high level visual features of the object but also information about its pose. Thus, a vector that presents only a few features that are related in a characteristic pose in that object is likely to yield to a right recognition.

Capsnets have attracted a great attention to real problems in short time. They have been successfully used in different computer vision tasks and fields such as hyperspectral image classification [10, 14], biomedicine [2, 9] and human 3D pose estimation from monocular images [11]. At the same time, researchers have explored different routing algorithms [4, 8, 15], trading-off the computational burden with the performance of this architecture [1, 3] and presented alternative architectures such as the Ladder Capsule Networks [7].

The contributions of this paper are: (i) a comparison of Capsnet architectures for recognition of garbage dumpster images taken as the operator found them in the street, (ii) the best performance so far in that real data set, and (iii) a discussion about the performance. The rest of the paper is as follows. We recap the most relevant facts about the Capsnet architecture in Section 2. The experimental set up is presented in Section 3. Results are discussed in Section 4. Finally we point out the conclusions and future work in Section 5.

2 Capsule architecture

Convolutional neurons (CNN) are able to learn convolution kernels that enhance different visual features of its input such as borders, corners and curves or even more complex patterns, depending on the kernel size. Thus, a layer of N convolutional neurons (that is N kernels) produces a feature map that consists of N images (channels), each one of them being the result of the convolution of the input image with the corresponding kernel. Therefore, a feature map collects different visual features. In deep convolutional architectures, these features are then pooled in order to preserve only the pixels with highest values. By doing so the resolution of the feature map is squeezed, usually to half height and half width, so the visual features are brought closer to repeat the process in the next convolution layer. After successive convolution and pooling layers the resulting feature map is used as the input to a classification network to carry out the recognition. This architecture strongly relies on the high level features extracted after the last convolution+pooling pair; taken as a single *flatten* vector, therefore missing spatial information. Thus, in order to be invariant to translations, rotations, flips, and different poses of the object, practitioners usually do data augmentation to have enough sample to generalize.

Capsules are an approach to obtain a representation that encodes both visual features and spatial information. To this end, the input is first preprocessed with a convolutional network to extract medium-level features. The resulting feature map is then fed into the capsnet, which consists of three main steps. For the sake of completeness we recap the main facts below (for further information we address the reader to [6] and [12]). From now on, we will assume that all the vectors are 1-column sized, unless it is said otherwise.

1. **Primary capsules** is the set of N d -dimensional vectors $\{\mathbf{u}_i\}$, with $i = 1, \dots, N$, obtained by chunking the feature map. The purpose of this first step is to group features that are close in the feature map into the same vector. This step is shown at the left side of Fig. 1, where the coloured area in the input image is representing an area related to the primary capsule coloured in the feature map. Notice also that the CNN

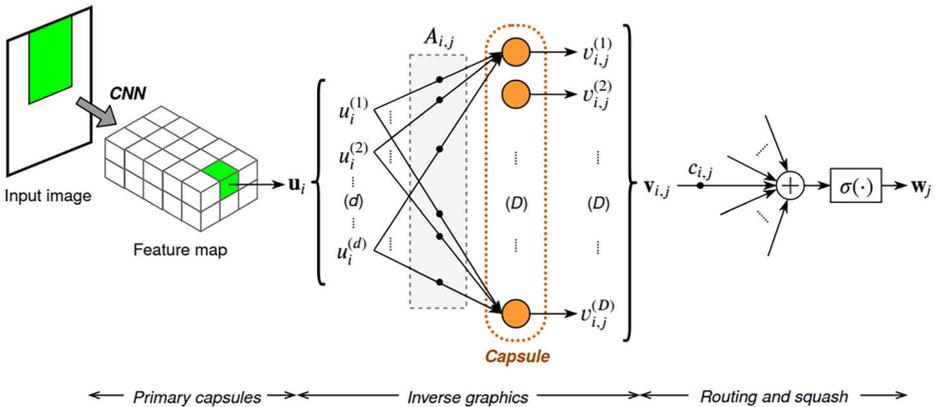


Fig. 1 Representation of a capsule as a group of neurons with linear activation, which is equivalent to a linear transformation given by the matrix $A_{i,j}$. The black dots representing weights of the synapses are the elements of $A_{i,j}$

that preprocesses the input has to be designed so that the dimensions of the feature map are compatible with an integer number of primary capsules, all of them equally sized. Additionally, there is no restriction to the their shape, but in this paper we only consider 2D primary capsules, also known as *low level capsules*.

2. **Inverse graphics** is a linear transformation A that maps each primary capsule into a D -dimensional space, with $D > d$. This transformation is done $N \times M$ times, each one of them with a different matrix A . Formally, $A_{i,j} \cdot \mathbf{u}_i = \mathbf{v}_{i,j}$, where $j = 1, \dots, M$, $A_{i,j} \in \mathbb{R}^{d \times D}$, and $\mathbf{v}_{i,j} \in \mathbb{R}^D$ are known as *activity vectors*. Therefore, each primary capsule \mathbf{u}_i is transformed into M activity vectors. The purpose of this step is to learn the features in the ‘real’ space where they live, for the initial capsules come from features obtained from a projection, i.e. the input image. Notice that each transformation $A_{i,j}$ is equivalent to a layer of fully connected neurons with linear activation that takes \mathbf{u}_i as input and produces the activity vector $\mathbf{v}_{i,j}$; hence the full set of transformations $\{A_{i,j}\}$ is simply learned with back-propagation as usual. Every $A_{i,j}$ is referred to as a ‘capsule of neurons’ in [12], giving the name to this architecture. This step is shown at the center of Fig. 1; where we have made explicit the elements of \mathbf{u}_i and $\mathbf{v}_{i,j}$ as $\mathbf{u} = [u_i^{(1)}, \dots, u_i^{(d)}]$ and $\mathbf{v}_{i,j} = [v_{i,j}^{(1)}, \dots, v_{i,j}^{(D)}]$ for the sake of clarity.
3. **Routing** is the process that linearly combines all activity vectors in the set $\{\mathbf{v}_{i,j}\}$, for $i = 1, \dots, N$ and a given j . Let $c_{i,j}$ be the coefficients of such a linear combination and let $\bar{\mathbf{v}}_j = \sum_{i=1}^N c_{i,j} \cdot \mathbf{v}_{i,j}$ be its resulting vector. The coefficients $c_{i,j}$ are obtained with the routing process. For instance, [12] presents the *dynamic routing*, that attempts to keep those activity vectors that most agree. During training, these coefficients are fitted three times in the feed-forward pass, and then remain fixed during back-propagation. Finally, let $\sigma(v) = \frac{\|v\|^2}{1+\|v\|^2} \cdot \frac{v}{\|v\|}$ be the squash function of a vector v ; which is used to normalize every $\bar{\mathbf{v}}_j$. Thus, let $\{\mathbf{w}_j\}$, with

$$\mathbf{w}_j = \sigma \left(\sum_{i=1}^N c_{i,j} \cdot \mathbf{v}_{i,j} \right), \tag{1}$$

be the set of final vectors, referred to as *final capsules* or *high level capsules* from now on. This step is shown at the right side of Fig. 1.

In summary, every capsule in the set $\{\mathbf{w}_j\}_{j=1,\dots,M}$ is a vector obtained as the result of activity vectors oriented in a similar direction. If there are as many capsules as classes, as in this paper, we can interpret the module of each one of these vectors as a degree of belonging to that class. Otherwise, we have to combine them into a fully connected neural network as it is customary for classification/regression tasks.

3 Experimental set up

In this section we introduce the data set, the different network architectures used in this paper, the data augmentation operations carried out for the analysis, and summarize the training and inference procedure.

3.1 Data set

We use the Ecoembes Dumpster Images Data set (EcoDID-2017), provided by the Spanish nonprofit company Ecoembes.² This data set provides a total of 27624 images of dumpsters, showing different conservation states, shapes and colors along with different lighting conditions as well as camera positions and *look at* angles. The resolution of the original images is 299×299 pixels but we rescale them down to 70×70 due to memory constraints. The images are given in seven folders (classes) according to the features shown in Fig. 2. An example of each type is displayed at Fig. 3.

3.2 Architectures

Following the architecture described in Section 2, we propose a baseline network, that will be referred to as A0, and four variations: A1 to A4. Architecture A0 consists of:

Preprocessing. It is a convolutional neural network as described in Table 1; where padding ‘same’ means that the output has the same width and height than the input, while ‘valid’ reduces both depending on the kernel size and the stride. The resulting feature map is a tensor in $\mathbb{R}^{16 \times 16 \times 32}$.

Capsules. We begin processing the feature map with an extra convolution layer of 256 kernels of size 9×9 , stride 2×2 , valid padding and no activation (a.k.a. *linear* or *identity*), which produces a tensor in $\mathbb{R}^{4 \times 4 \times 256}$. Primary capsules are obtained chunking this tensor into a set of 512 vectors of 8 elements, denoted as $\{\mathbf{u}_i \in \mathbb{R}^8\}$, for $i = 1, \dots, 512$, in Fig. 1. The inverse graphics are carried out with matrices $A_{i,j} \in \mathbb{R}^{8 \times 16}$, for $i = 1, \dots, 512$ and $j = 1, \dots, 7$, as many as different classes for each primary capsule; resulting a set of 3584 activity vectors $\{\mathbf{v}_{i,j} \in \mathbb{R}^{16}\}$. Next, we apply the Dynamic Routing proposed in [12] to $\{\mathbf{v}_{i,j}\}$ and the *squash* function to the result of it, finally obtaining the set of *high level* capsules $\{\mathbf{w}_j\}$.

Header. We use two headers at the same time. The main one is for predicting the class of the input image. Specifically, the class predicted is the one associated to the high level capsule with greatest module. The other header attempts to reconstruct the input image and consists of three dense layers of 512, 1024 and 14700 successively. The output is reshaped to $70 \times 70 \times 3$ to match the original image resolution.

²The dataset must be requested to the corresponding author

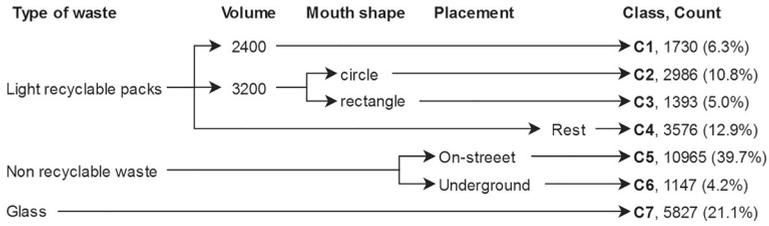


Fig. 2 Taxonomy and number of images contained in each folder of EcoDID-2017

3.2.1 Architectural modifications

We have also tested four variations of A0; one with a different header and the rest (A1, A2, A3 and A4) with different preprocessing stage. We stress that there are 512 primary capsules, each being an array of 8 elements in all of them, so the capsule stage remains invariant. Thus, we can assess the impact of the double loss function in the training of the capsule network and find the best feature map for the classification task.

- A1: It is the architecture A0 without the reconstruction header.
- A2: It is the architecture A1, replacing the preprocessing stage shown in Table 1 by the one shown in Table 2. Notice that the change only affects to the preprocessing but the output of this stage remains in $\mathbb{R}^{16 \times 16 \times 32}$.
- A3: It is the architecture A2, but increasing the number of filters N from 32 to 128 in both layers. Hence, the output of the preprocessing stage is a tensor in $\mathbb{R}^{16 \times 16 \times 128}$.
- A4: It is the architecture A3, but with filters of size 5×5 instead of 3×3 . This modification produces a tensor in $\mathbb{R}^{15 \times 15 \times 128}$ as output of the preprocessing stage.



Fig. 3 A sample of each dumpster type in the data set

Table 1 Layers of the preprocessing stage in architecture A0

| Order | Layer | N | size | Strides | Padding | Activation |
|-------|---------|----|--------------|--------------|---------|------------|
| 1 | conv 2D | 32 | 3×3 | 1×1 | valid | ReLU |
| 2 | maxpool | | 2×2 | 1×1 | same | |
| 3 | conv 2D | 32 | 3×3 | 1×1 | valid | ReLU |
| 4 | maxpool | | 2×2 | 1×1 | same | |

3.3 Data augmentation

In this work we have tested four transformations, which are represented over an original image in Fig. 4.

Jit: Random color jitter in a range [0%, 10%] in brightness, contrast, saturation and hue.

Crop: Random crop at a random position in the range [0, 7] from every corner.

Rot: Random rotation in the range [−10, 10] degrees.

Flip: Horizontally flip the original image randomly with 0.5 probability.

In the training pipeline of architectures A0 to A4, every image fed into the network has a probability of being modified. If so, then Flip and Crop are applied. After having evaluated all the architectures, we carried out additional experiments with eleven other transformation pipelines that include also Rotation and Jitter operations, but only with A4 (see details in Section 4).

3.4 Loss function

All the architectures produce as many high-level capsules (that live in a vector space) as there are different labels, and the predicted label is that one with greatest module. Therefore, there is not dense neuronal network acting as a predictor header. However, architecture A0 includes a reconstruction header. For this reason the loss function has two terms for A0 and only one term for the rest, unless the reconstruction header is appended (see Ablation study).

To train the predictions, we use the Margin loss [12], whereas to train the reconstruction header we include the Mean Squared Error (MSE). Thus, the total loss L is a weighted combination if both are necessary. In that case, given the order of magnitude of both, we finally choose $L = 5 \cdot 10^{-4} \cdot \text{MarginLoss} + \text{MSE}$. Otherwise, the loss is the Margin Loss, without the weight.

3.5 Training and inference

For the sake of completeness, we summarize here how to train the proposed neural network and how to make inference.

Table 2 Layers of the preprocessing stage in architecture A2

| Order | Layer | N | size | Strides | Padding | Activation |
|-------|-------|----|--------------|--------------|---------|------------|
| 1 | conv | 32 | 3×3 | 2×2 | valid | ReLU |
| 2 | conv | 32 | 3×3 | 2×2 | valid | ReLU |

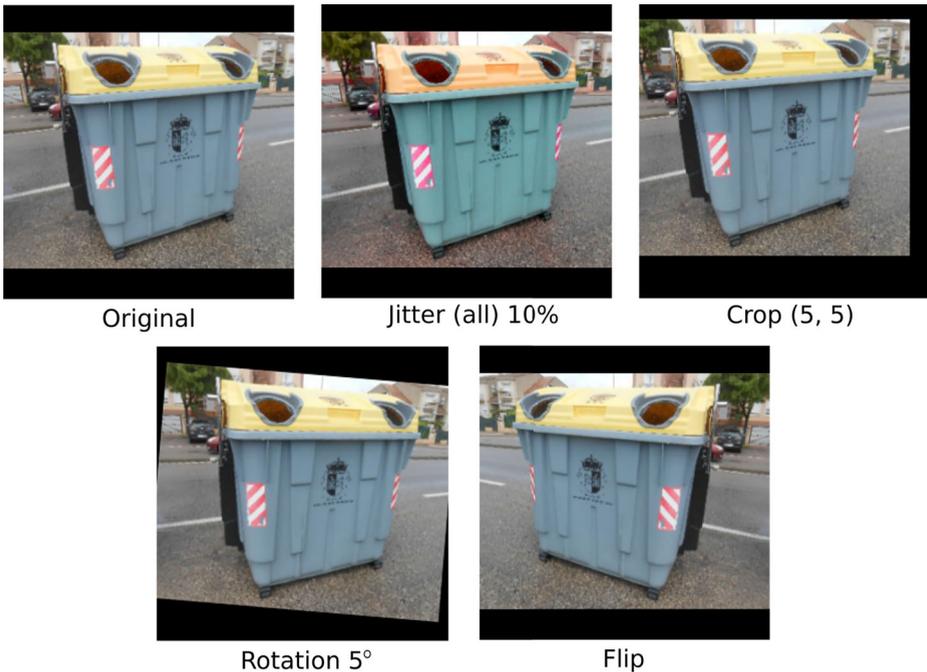


Fig. 4 A sample of each operation applied during data augmentation.

First of all, the data set has to be split in two sets: one for learning and another of testing. Yet, in order to prevent overfitting, the first set is split again in two. In this paper the data set has been split in Training (60%), Validation (20%) and Test (20%); all of them keeping the same the ratio of images per class than the whole data set.

Training a neural network such as the one presented here consists of an iterative process with three steps. 1) *Feed-forward*; training images are fed into the network, which acts like a parametric and derivable function that produces the predicted label of these images. 2) *Loss*; since the true label is known it is possible to measure the discrepancy of the predicted label by means of a loss function that has to be minimized. 3) *Back-propagation*; if the loss function is derivable, then the parameters of the neural network can be updated via gradient descent methods. Every complete iteration gets new images from the training set up to all of it has been used; although it is customary to start over, beginning a new *epoch*.

Each one of these steps is carried out differently from one problem to another. In the feed-forward, images can go in batches instead of one by one. Specifically, we use a batch size of 16 images, and each one of them is processed by the augmentation strategy as discussed above. Regarding the loss function, we gave the details in Section 3.4. Finally, gradient descent methods are controlled by several hyperparameters. In our paper we use Adam optimizer with learning rate = 0.001 and learning rate decay = 0.99 in each epoch.

Inference is just a feed-forward step with new images, which are input one by one because we are interested in their predicted label; for this reason it is much faster than training. Inference is done with the validation set in order to evaluate how the training is progressing; and with the test set to assess the accuracy of the predictions with images never fed during training, mimicking what will happen once the neural network is on production.

Currently, Tensorflow and PyTorch are the two most popular software libraries for implementing, training and testing neural networks. Both provide methods that make invisible the gradient descent algorithms so one can focus only on designing the architecture and programming the loss function if it is not a standard one, as in this paper. Specifically, we have used PyTorch, and the code is publicly available.³

4 Experimental analysis

All the experiments were run on a Intel Core i7-8750HQ (2.2Ghz, 9MB) with 16GB RAM DDR4 and a GPU GTX 1070 8GB GDDR5, and training is always performed for 100 epochs.

We have carried out four analysis; the first one is for choosing the architecture out of the five alternatives, whereas the rest are to assess its goodness.

4.1 Architecture comparison

The state-of-the-art with EcoDID-2017 is due to [11], with a Google Inception V3 network, pretrained in Imagenet, and consisting of 23.8 million parameters. We use this network as baseline to compare with capsule network architectures A0 to A4 in three axis: accuracy, millions of parameters and training time per epoch (TTPE), measured in seconds. Results are shown in Table 3 and visualized in Fig. 5-Left. The millions of parameters are represented in the size of the balls, the dotted line is the accuracy attained in test and the solid line is the TTPE.

The first relevant fact is that a drop in the number of parameters has an impact on the TTPE, which is expected since there is much less computational load. A second remark is that A2, A3 and A4 attain a better accuracy than the rest. Moreover, the accuracy increases as we modify the architectures, up to A4 that has the best performance, with a 95.35% accuracy on the test data set. Finally, we also stress that none of the capsule network architectures tried benefits from transfer learning, while Google Inception V3 did.

For these reasons, architecture A4 has been chosen as the final classification network. In order to assess its goodness, we carry out an ablation study, a data augmentation strategy comparison and a training size sensitivity analysis.

4.2 Ablation study

We make seven modifications over architecture A4, which are summarized in Table 4, where the first row corresponds to A4, and the first column contains the code for each change.

First, we add the reconstruction header again (row +R), that was removed from A1 on. This extra header greatly increases the number of parameters and the TTPE; but the reduces the accuracy 0.67 percent points. This leads us to think that the variety of containers within any class makes the reconstruction much harder; so the classification task is dominating the gradient descent and the reconstruction is slowing it down.

Second, we remove the Crop-Flip data augmentation (row -DA). This modification produces the worst accuracy, 3.13 percent points down with respect to A4.

³<https://github.com/capo-urjc/Capsnet-for-dumpsters>

Table 3 Comparison between our proposed models (A1-A4) and state-of-the-art (SotA) architectures Inception V3 [11] and the original CapsNet (A0). **Param.** is the millions of parameters, **Acc.** is the the percentage of accuracy, **TTPE** is the seconds of training time per epoch. The TTPE of Inception V3 was not reported in [11]

| Architecture | Param.($\times 10^6$) | Acc.(%) | TTPE(s.) |
|-----------------|-------------------------|---------|----------|
| Incep.V3 (SotA) | 23.8 | 93.00 | n.r. |
| A0 - Capsnet | 38.8 | 90.88 | 325.74 |
| A1 | 23.1 | 92.20 | 315.54 |
| A2 | 1.1 | 93.12 | 13.95 |
| A3 | 3.2 | 94.52 | 15.73 |
| A4 | 3.5 | 95.35 | 15.80 |

Third, we remove one of the two convolutional layers in the preprocessing stage (row 1CNN). As a consequence, the output of the preprocessing stage is a tensor in $\mathbb{R}^{34 \times 34 \times 32}$, and the convolution layer at the beginning of the capsule network transforms it into $\mathbb{R}^{34 \times 34 \times 32}$. Since we keep the primary capsules in \mathbb{R}^8 the number of them, as well as the number of matrices $A_{i,j}$, increases from 512 to 5408. This explains the great increase in the number of parameters. On the other hand, missing that convolutional layer accounts for poorer visual features, which has an impact on the accuracy, losing 2.72 percent points with respect to A4.

Forth, we try with more and less filters in the two convolutional layers of A4 preprocessing stage. Specifically, we try 64 and 256 filters (rows 64F and 256F respectively); which is half and double the number of filters respectively. In the same line, we also try two different kernel sizes, one smaller and one larger: 3×3 and 9×9 (last two rows).

These results are depicted in Fig. 5-Right. Notice that the right-most ball in Fig. 5-Left is the same than the left-most ball in Fig. 5-Right, both corresponding to A4; showing to have the greatest accuracy value being one of the smallest in terms of millions of parameters.

4.3 Data augmentation pipeline comparison

All the architectures A0 to A4 had a data augmentation pipeline consisting of Crop and Flip operations as described above. Once A4 has been selected, we try all the variations listed in Table 5 in decreasing order of accuracy. Results are also depicted in Fig. 6. It can be clearly noticed that the Jittering operation increases the TTPE because it is pixel-wise. Besides the

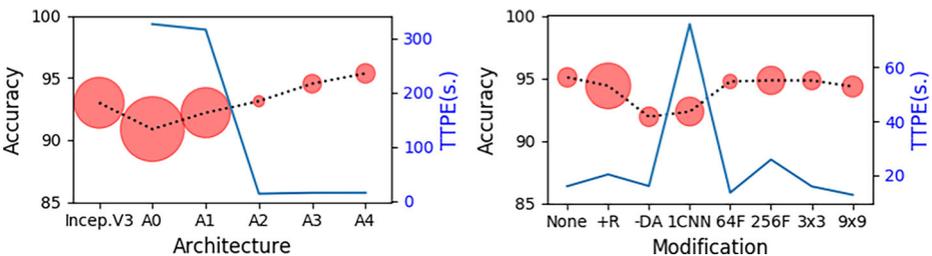


Fig. 5 (Left) Visualization of Table 3. (Right) Visualization of Table 4. In both, the ball size is proportional to the millions of parameters of the architecture; the dotted line is the accuracy (%), left Y-axis and the solid line is the TTPE (s., right Y-axis)

Table 4 Ablation study

| Modif. | Param. ($\times 10^6$) | Acc.(%) | Acc. down | TTPE(s.) |
|--------|--------------------------|---------|-----------|----------|
| None | 3.5 | 95.06 | 0.00 | 15.80 |
| +R | 19.2 | 94.39 | 0.67 | 20.22 |
| −DA | 3.5 | 91.93 | 3.13 | 15.86 |
| 1CNN | 7.5 | 92.34 | 2.72 | 76.11 |
| 64F | 1.9 | 94.73 | 0.33 | 13.40 |
| 256F | 7.4 | 94.82 | 0.24 | 25.75 |
| 3 × 3 | 3.2 | 94.81 | 0.25 | 15.73 |
| 9 × 9 | 4.1 | 94.33 | 0.73 | 12.59 |

accuracy is down more than one percent point only when Cropping is not used. Therefore, the initial choice of Crop and Flip remains the best trade-off between accuracy and TTPE.

4.4 Training size sensitivity analysis

Finally, we consider reducing the number of training images by using the 80%, 60%, 40% and 20% of the training set. We show the confusion matrix with the full training set in Table 6 (above) and in the left-most panel of Fig. 7. Table 6 (above) has not absolute values but the percentage with respect to the number of images in the corresponding class. The rest of panels depict the cell-wise difference between that one with the confusion matrix obtained using less training examples. Thus, the darker a diagonal cell the more images of that class mislabeled. Qualitatively, the confusion matrix diagonal gets darker as the training set reduces, meaning that there are less images correctly labeled. Specifically, Table 6 (below) shows the percent point difference between the diagonal of the confusion matrix using 100% training set and the corresponding reduced one. Classes C4, C5 and C7 are quite robust which is probably indicating a greater inter-similarity between the images inside each class. Nevertheless, in all the cases using only 1 out of 5 images per class clearly degrades the performance.

Table 5 Data augmentation ablation study

| Transformation pipeline | Accuracy | Acc. down | TTPE (s.) |
|---------------------------------|----------|-----------|-----------|
| <i>Crop-Flip</i> (CF) | 95.06 | 0 | 15.80 |
| <i>Rot-Crop</i> (RC) | 94.97 | 0.09 | 15.90 |
| <i>Crop-Jit-Flip</i> (CJF) | 94.91 | 0.15 | 20.42 |
| <i>Rot-Crop-Jit-Flip</i> (RCJF) | 94.91 | 0.15 | 20.68 |
| <i>Rot-Crop-Jit</i> (RCJ) | 94.77 | 0.29 | 20.70 |
| <i>Rot-Crop-Flip</i> (RCF) | 94.75 | 0.31 | 15.85 |
| <i>Jit-Crop</i> (JC) | 94.21 | 0.85 | 19.98 |
| <i>Flip-Rot</i> (FR) | 93.85 | 1.21 | 15.90 |
| <i>Rot-Jit-Flip</i> (RJF) | 93.48 | 1.58 | 20.48 |
| <i>Jit-Rot</i> (JR) | 92.40 | 2.66 | 20.21 |
| <i>Jit-Flip</i> (JF) | 92.25 | 2.81 | 20.01 |
| <i>None</i> | 91.93 | 3.13 | 15.86 |

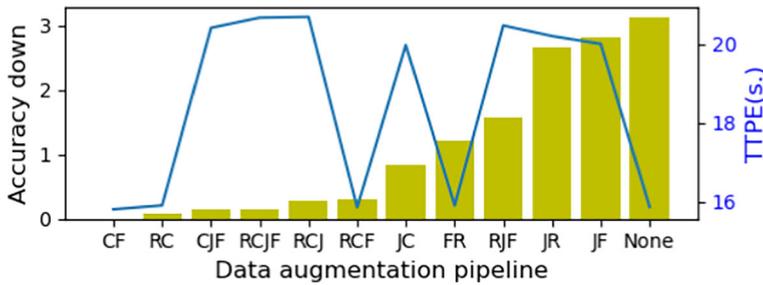


Fig. 6 Loss in accuracy with respect to Crop-Flip augmentation strategy (left vertical axis and bars) and Training time per epoch (right vertical axis and line) vs. the rest of augmentation strategies listed in Table 5

Additionally, we have carried out a binary classification with a One-vs-Rest strategy for each class. Results are depicted in Fig. 8, where the X-axis is the training data set size and the Y-axis is the Precision, Recall and F1-score respectively. The three metrics are quite stable as the training data set size decreases. We remark that the more images per class, the lower these metrics are, which reflects the difficulty of the task increases with the size of the sample, mainly due to the variability mentioned above.

4.5 Discussion

After experiments, A4 together with Crop and Flip augmentation functions proved to be the best option; successfully outperforming the previous best result with 83% fewer parameters; but it still remains to discuss whether the results are compatible with the expectations.

Firstly, we removed the reconstruction header of A0 to get A1. The purpose of this header is to regularize the learning process; but the amount of parameters that introduces to generate every pixel makes A0 even bigger than Google Inception V3 and forces to design a loss function that mixes both the reconstruction and the classification task, which causes the

Table 6 (Above) Confusion matrix with all training data. Rows are true labels and columns are predicted labels. Values are the percentage with respect to the number of images in that class. (Below) Percent point difference of the correctly labeled images for each class as the training data set size decreases

| | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|------|-------|-------|-------|-------|-------|-------|-------|
| C1 | 90.69 | 3.47 | 0.92 | 4.45 | 0.40 | 0.00 | 0.06 |
| C2 | 2.68 | 92.37 | 0.77 | 3.88 | 0.23 | 0.03 | 0.03 |
| C3 | 1.58 | 1.29 | 89.96 | 6.95 | 0.07 | 0.07 | 0.07 |
| C4 | 2.65 | 2.65 | 5.34 | 85.50 | 1.68 | 1.26 | 0.92 |
| C5 | 0.05 | 0.00 | 0.01 | 0.33 | 99.08 | 0.08 | 0.45 |
| C6 | 0.00 | 0.00 | 0.17 | 0.44 | 0.61 | 94.09 | 4.70 |
| C7 | 0.02 | 0.00 | 0.02 | 0.49 | 0.86 | 1.70 | 96.93 |
| Size | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
| 80% | 1.7 | 1.7 | 3.8 | 2.3 | 0.3 | 4.1 | 0.5 |
| 60% | 1.7 | 1.7 | 3.8 | 2.3 | 0.3 | 4.1 | 0.5 |
| 40% | 5.5 | 4.1 | 4.6 | 2.2 | 0.3 | 5.7 | 0.7 |
| 20% | 12.2 | 7.5 | 6.4 | 6.3 | 1.2 | 12.1 | 2.7 |

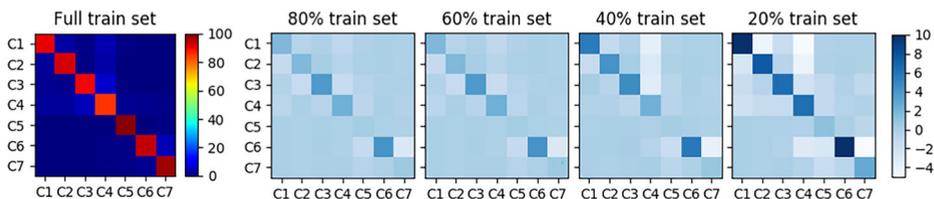


Fig. 7 (Left) Confusion matrix with validation data using the full training set. (Others) Difference between the confusion matrix with full training set and with a reduced one

accuracy to drop below the state-of-the-art. Thus, removing the header brings the following consequences: 1) the loss function consists only of the so called Margin Loss, 2) there is no classification header because the estimated label is the high level capsule (i.e. the outcome of the capsule network) with greatest module, 3) reduces the size of the whole network, and 4) we recover almost the same accuracy than the Google Inception V3. In the ablation study we confirm these effects, since we connect the reconstruction header again on A4 and show that the accuracy decreases again.

Next, in order to keep on reducing the number of parameters, we removed the Maxpool layers, doing both the feature extraction and the resolution reduction with the convolution stride twice as large as in A1. The resulting A2 architecture has $\times 20$ less parameters than A1, and its accuracy already slightly surpasses the state-of-the-art. Arguably, a much smaller parameter space has two advantages: fewer local optima and more expressive feature representation. Indeed, we exploit this latter idea to improve the architecture by adding more feature maps, from 32 in A2 to 128 in A3. Thus, the more visual features the CNN preprocessing extracts, the more initial capsules are available for the capsule network stage to model the hierarchical relationships between them. Finally, we changed the 3×3 kernels for 5×5 kernels, keeping the same number of feature maps. Consequently, the visual features extracted by A4 are of higher level than those of A3.

All in all, the A4 architecture has a compact and efficient CNN that extracts the visual features that best fit how the capsule network processes its inputs. Each new version, from A0 to A4, is designed to achieve better performance, and experiments confirm this goal.

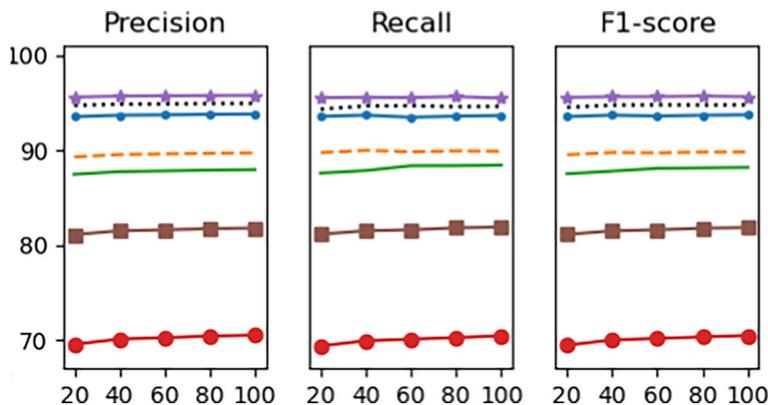


Fig. 8 Precision, Recall and F1-score for each class vs. the rest as the training data set size is reduced. Lines are: C1(solid-dots, blue), C2(dashed, orange), C3(dotted, black), C4(solid, green), C5(solid-circles, red), C6(solid-stars, magenta) and C7(solid-squares, brown)

Since we are using a two-layer CNN followed by one capsnet, another relevant discussion is why not using deeper architectures. More convolutional layers account for higher level visual features. But Capsule networks are efficient with mid-level features because they are designed to “look” at different parts of the image and select those that agree. If the receptive fields that are looked at are very small or large, this effect is diminished. In the ablation study we show that more and less feature maps, as well as smaller and larger convolutional kernels, attain slightly worst performance.

The use of another capsule network would introduce an excessive computational cost due to the dynamic routing algorithm; which consists of a clustering-like process with all the activity vectors, that has been carried out 3 times. Finally, the CNN followed by the capsnet is expected to produce as many high-level capsules as there are different labels. In other words, each output (high-level capsule) is a vector representing the input image according to the label corresponding to that output. The higher the module of the vector, the better the match; therefore, the estimated label is the one with the highest module. For this reason we do not add a classification header consisting of a dense neural network with a softmax activation function.

5 Conclusions and future work

In this paper we present a solution for classifying images of dumpsters. The task is far from trivial, even with the current techniques, due to the problems induced by a *real* data set, as it is collected and served by a company or client. We use one of the most promising deep learning architectures, Capsule networks, and present a systematic study for boosting the performance above the state-of-the-art in the given data set. Additionally, we report a comprehensive experimental analysis of the solution that includes an ablation study, a comparison of different augmentation data pipelines and a study of the sensitivity to the size of the training data set. All the experiments have more than one metric in order to evaluate possible trade-offs. The solution proposed, architecture A4 with a pipeline of Crop and Flip operations, has proved to be the best choice, with an accuracy of 95.35%.

The EcoDID-2017 database of dumpster images provides a large variety of labels describing the conservation status such as *broken*, *burned*, *deformed*, among other relevant defects. A future research line is to achieve automatic dumpster status labeling. This application is highly demanded by the company that maintains the database; and it is also very interesting as a smart maintenance service concept for smart cities. As dumpsters are emptied regularly by garbage trucks, an on-board vision-based system running this application would update the dumpster status on the fly. Additionally, we plan to extend these smart diagnosis system to consider different urban furniture.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This research has been supported by the Spanish Government research funding RTI2018-098743-B-I00 (MICINN/FEDER) and the Comunidad de Madrid research funding grant Y2018/EMT-5062. We also would like to acknowledge Ecoembes for providing the EcoDID-2017 database of dumpster images for this work.

Declarations

Conflict of Interests All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Ahmed K, Torresani L (2019) STAR-caps: Capsule networks with straight-through attentive routing. In: Advances in neural information processing systems, vol 32, pp 9101–9110
2. Baydilli YY, Atila U (2020) Classification of white blood cells using capsule networks. *Comput Med Imaging Graph* 80:101699
3. Ding X, Wang N, Gao X, Li J, Wang X (2019) Group reconstruction and max-pooling residual capsule network. In: Proc of the 28th Int Joint Conf on artificial intelligence (IJCAI)
4. Hahn T, Pyeon M, Kim G (2019) Self-routing capsule networks. In: Advances in neural information processing systems, vol 32, pp 7658–7667
5. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: IEEE Conf on computer vision and pattern recognition (CVPR), pp 770–778
6. Hinton GE, Sabour S, Frosst N (2018) Matrix capsules with EM routing. In: Int Conf on learning representations (ICLR)
7. Jeong T, Lee Y, Kim H (2019) Ladder capsule network. In: Proc of the 36th int conf on machine learning (ICML), Long Beach, California, USA, vol 97, pp 3071–3079
8. Lenssen JE, Fey M, Libuschewski P (2018) Group equivariant capsule networks. In: Advances in neural information processing systems, vol 31, pp 8844–8853
9. Mobiny A, Lu H, Nguyen HV, Roysam B, Varadarajan N (2020) Automated classification of apoptosis in phase contrast microscopy using capsule network. *IEEE Trans on Medical Imaging* 39(1):1–10
10. Paoletti ME, Haut JM, Fernandez-Beltran R, Plaza J, Plaza A, Li J, Pla F (2019) Capsule networks for hyperspectral image classification. *IEEE Trans on Geoscience and Remote Sensing* 57(4):2145–2160
11. Ramírez I, Cuesta-Infante A, Schiavi E, Pantrigo JJ (2020) Convolutional neural networks for computer vision-based detection and recognition of dumpsters. *Neural Comput & Applic* 32(17):13203–13211
12. Sabour S, Frosst N, Hinton GE (2017) Dynamic routing between capsules. In: Advances in neural information processing systems 30
13. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2016) Rethinking the inception architecture for computer vision. In: IEEE conf on computer vision and pattern recognition (CVPR), pp 2818–2826
14. Xu Q, Wang DY, Luo B (2020) Faster multiscale capsule network with octave convolution for hyperspectral image classification. *IEEE Geosci Remote Sens Lett*, pp 1–5
15. Zhang L, Edraki M, Qi GJ (2018) CapProNet: Deep feature learning via orthogonal projections onto capsule subspaces. In: Advances in neural information processing systems 31, pp 5814–5823

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.