A comparison between CapsNet and traditional Convolutional Neural Networks on the Google Landmark Recognition Challenge

Giulia Milan - Enrico Postolov - Davide Taddei
DAUIN - Politecnico di Torino
s264976 - s262745 - s267612
<studentID>@studenti.polito.it

Abstract

In the Google Landmark Recognition Challenge the goal is to recognize famous landmarks from images. Some solutions have already been done using existing CNNs. We want to solve this challenge with CapsNet, a new kind of network. CapsNet is based on the dynamic routing between capsules. This network could find practical applications in deep learning, overcoming some of the limitations of CNNs, because those capsules can preserve and model hierarchical relationships better than Convolutional Neural Networks.

1. Introduction

In this paper we want to compare a recent neural network, called CapsNet, with the existing Convolutional Neural Networks on a subset of the Google Landmark Recognition Challenge dataset. We took into account the following CNNs: VGG16, AlexNet, DenseNet161 and ResNet50. We evaluated the accuracies obtained by those networks and the training and test execution times for each of them, in order to compare their performances. This work led us to reflect on possible future experiments on CapsNet network, that will be described in the last section.

2. Problem

The Google Landmark Recognition Challenge [12] consists in recognizing landmarks in images, predicting landmark labels directly from image pixels. Implementations already exist for this purpose, using different CNNs.

CapsNet is a new network that has been mainly used in research activities and, until now, has been applied only on simple and small image datasets, such as MNIST [16]. A CapsNet implementation for the Google Landmark dataset does not exist according to our current knowledge. It is interesting to compare the performance of CapsNet in relation to the other CNNs.

According to the implementation of the Google Landmark Recognition Challenge in [6], VGG16 turned out to be the best performing network among VGG16, Inception v3, ResNet and DenseNet. For having the chance to compare CapsNet's performance with the most common CNNs, we decided to use as benchmarks the following networks, which are the most relevant in the computer vision area:

- VGG16 [11]: it is the best performing network according to [6] for the Google Landmark Recognition Challenge.
- AlexNet [2]: it competed in the ImageNet Large Scale Visual Recognition Challenge in 2012 and according to [4] it is considered one of the most influential papers published in computer vision.
- DenseNet161 [9]: This paper in 2017's CVPR got Best Paper Award with over 2000 citations, so this is an important network to take into account.
- ResNet50 [10]: it made it possible to train up to hundreds of layers and still achieves compelling performance, with respect to the previous state-of-art CNNs, such as AlexNet.

3. Context and data

The dataset that will be used is the Google Landmarks dataset, made available by [8].

The Google Landmarks Dataset, provided by Google, contains around 15,000 classes, with a number of images near to 1,225,000.

Since training the networks takes a lot of time and computational resources, especially for the CapsNet, we decided not to take all the data from the dataset. We filtered it by taking 50 classes out of 15,000, and for each class we retrieved a number of images equal or almost equal to 100. We took this decision because the Google Landmarks dataset has unbalanced classes in terms of number

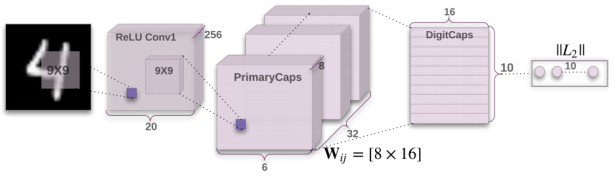

Figure 1. CapsNet implementation for MNIST dataset. [14]

of images, in fact all the classes have a number of images between 50 and 100, but, for example, there are very few classes containing more than 1000 images. Thus, we thought that the best approach, in terms of distribution, was to take the first 50 classes with 100 (or nearly 100) images.

We used an already provided template to download images from the Google Landmarks dataset provided by Tobias Weyand [5]. As specified in Kaggle, some images may not be available at download time, so the script skips the images which are not available.

To solve this challenge, CNNs such as VGG and ResNet have already been used in other implementations. The problem with this kind of neural networks is that, in the most cases, meaningful information regarding spatial relationships among the features could be lost: this influences a lot the predictions of the model [13]. For example, those information could regard size, orientation and perspective. Exactly for this reason, the neural network we decided to adopt, in order to have a performance comparison, is CapsNet, a very recent capsule network [14].

A capsule network is an artificial neural network architecture developed by Geoffrey Hinton. In the capsule network, images are processed using an approach based on equivariant mapping and the mapping of the hierarchy of parts. Equivariant mapping enables the preservation of position and pose information, that is one of the most interesting properties of the CapsNet and a significant difference with respect to traditional Convolutional Neural Networks.

In details, CapsNet has the peculiarity of better preserving the orientation of the objects, being thus able to overcome the well known "Picasso problem" in image recognition, in which there are images showing the right component but without the right spatial relationships. An example is when in a "face" the location of the eye and of the ear are swapped. While viewpoint changes have nonlinear effects at the pixel level, they have linear effects at the object or part level, and CapsNets exploits this. This is like inverting the rendering of an object with several parts.

Some CapsNet implementations already exist. We decided to start from the one developed using the PyTorch framework by Jackie Loong [3]. Of course, we modified this implementation to make it more suitable to our needs, as we will explain in the next section.

4. Methods

CapsNet is a new Artificial Neural Network (ANN) which is based on dynamic routing between capsules.

A capsule can be considered as a group of neurons whose activity vector represents the parameters related to the instantiation of a specific type of entity, such as an object or an object part [15]. Capsules model a hierarchical relationship and they are structured in a way that simulate biological neurons.

The capsule approach only considers the existence of the object in the image around a specific location. It does not care about the direction of the object or spatial relations. A capsule is a vector. Capsules denote the features of the object and its probability. These features can include parameters such as "pose" (size, position, orientation), velocity, deformation and so on [1].

As previously written, some CapsNet implementations already exist and they focus mainly on classifying digits from the MNIST dataset [16]. A representation of the traditional CapsNet that can be applied on the MNIST dataset can be found in Figure 1

In the following, the most significant modifications to the existing implementation of CapsNet are presented:

- The code related to the download of the images from the Google Landmarks dataset has been added, available in [5]. The script is able, in an efficient way, to detect if an image has been already downloaded or if it is not available. The downloaded images have a size of 1600x1200 pixels. As first thing, they are automatically resized to 224x224 pixels, in order to give them as input to the CNNs we evaluated. In this way, the script for downloading the images is the same for the CapsNet and for the other CNNs implementations.

- In the transformations applied to train and test splits, data have been modified in order to normalize picture with standard mean and variance equal to 0.5 for all RGB channels of the images. Moreover, specifically for the CapsNet, the images have been resized to 28x28 pixels in order to be processed properly by the CapsNet without overloading our limited hardware resources. In fact, using bigger images causes the RAM to be overloaded and Google Colab, which is the platform we used to run our networks, to not work properly and to stop its execution. In the project proposal we proposed to add some convolutional layers to reduce the initial size of input images of 224x224. We tried to do that, but we did not notice any significant advantage. Moreover, the more convolutional layers are added to the network, the more it suffers the same limitations of the classic CNNs: particularly, the one of being sensible to spatial relationships. With 28x28 input images we obtained relevant results, that will be

```
  (net): Sequential(
    (0): Conv2d(3, 256, kernel_size=(9, 9), stride=(1, 1))
    (1): ReLU(inplace=True)
    (2): Caps(
      (subnet0): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet1): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet2): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet3): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet4): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet5): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet6): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet7): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
      (subnet8): Conv2d(256, 32, kernel_size=(9, 9), stride=(2, 2))
    )
    (3): Route()
  )
)
```

Figure 2. Final CapsNet implementation with modifications for the Google Landmarks dataset.

discussed later in this paper, so we decided to stick to this input dimension.

- Since the MNIST dataset is composed by grayscale images while landmarks are RGB images, we modified the first convolutional layer of the CapsNet in order to be able to manage 3 input channels instead of only 1.

- Moreover, while the traditional CapsNet implementation deals with 8 capsules, we worked with a total of 9 capsules. We took this decision because, after multiple executions, we found out that the results achieved were better. We did not find any significant improvement adding 2 capsules tough, very probably because of the hardware limitations pointed out previously.

- To achieve more flexibility to the usage of the network implemented, we defined a NUM_CLASSES parameter to set the number of outputs of the network corresponding to the number of classes of the filtered input dataset.

- We split the filtered input dataset into train and test sets, respectively with percentages of 80% and 20% with respect to the one provided in input. Of course, this holds also for the other CNNs we used.

- As batch size, we used a value equal to 32, for the CapsNet. The original CapsNet implementation uses a higher value (150), but after several trials, we noted that a smaller value was more suitable for our goal, without consuming too many resources.

In Figure 2 the CapsNet custom implementation with modification for Google Landmarks Dataset is described.

For the other CNNs, we used the same download code and the same training and test sets of the CapsNet. For not pretrained CNNs, we normalized both mean and variance to 0.5 as we did in the CapsNet implementation, while for the pretrained networks we normalized the mean and variance using the values suggested by PyTorch, since the pretrain is performed over ImageNet.

It is important to notice that an existing implementation of CapsNet pretrained on ImageNet does not currently exist, as far as we know. However, we decided to run the CNNs both in pretrained and not pretrained mode. This has been done because if the CapsNet obtains reasonable results without being previously trained, it could be possible that a pretrained version of it on ImageNet would make CapsNet competitive with respect to the other CNNs.

5. Evaluation

In the following sections the methods we used in order to evaluate the performance of all the networks and the results obtained for each of them will be described.

5.1. Methods

We performed evaluation dividing the initial dataset into training and test splits, with 20% of images belonging to the test set and the remaining 80% belonging to the training set. We divided the dataset so that on average all classes are equally represented by images in the test set.

For each epoch, the accuracy and loss on training and test sets are evaluated. At the end of the training phase, each model is tested on the training set, evaluating the final accuracy value. Those outcomes will be evaluated for CapsNet and for all the other CNNs. We did not perform a validation step.

All networks have been evaluated using the Google Colab notebook. The public repository with available scripts can be found in [7]. We used the same environment for all the executions, in order to have reliable and consistent results.

For a fair comparison with CapsNet, we used the not pretrained CNNs to compare performances and accuracies. Results for a pretrained CapsNet are not available since CapsNet has not been trained on the ImageNet dataset yet. Instead we executed the other networks also exploiting transfer learning in order to obtain additional accuracies and information. Our hypothesis is that if the results obtained with the current implementation of CapsNet are comparable to the ones obtained with the not pretrained CNNs, it's possible that a pretrained CapsNet could have comparable results to the other pretrained CNNs.

5.2. Results

To evalutate the results, accuracies on the test set after 30 epochs have been computed. The learning rate for the CNNs is 1e-3, while for the CapsNet we found out that the most interesting results were achieved using a learning rate of 1e-2 and a batch size value of 32.

In Table 1, accuracies reached with pretrained CNNs, not pretrained CNNs and CapsNet model are shown.

Network	Not pretrained	Pretrained
CapsNet	0.7188	NA
VGG16	0.7874	0.9862
AlexNet	0.3954	0.9437
DenseNet161	0.8908	0.9931
ResNet50	0.8023	0.9897

Table 1. Accuracies evaluated both for pretrained and not pretrained networks. Pretrained CapsNet model is not available.

Network	Training time	Test time	Total time
CapsNet	3:03:33	0:00:01	3:03:34
VGG16	0:33:19	0:00:07	0:33:26
AlexNet	0:06:29	0:00:04	0:06:33
DenseNet161	0:42:40	0:00:08	0:42:48
ResNet50	0:11:06	0:00:04	0:11:10

Table 2. Training and testing time for not pretrained networks (h:mm:ss).

Looking at not pretrained models, it can be noted that the best performing model results to be the DenseNet161, with an accuracy of 89%, while the CapsNet reaches a result near to 72%, which can be almost compared with the one obtained by the VGG16. CapsNet accuracy is also significantly greater than the AlexNet one, of 40%. For what concerns the pretrained models (in which the CapsNet is not present), the best accuracy is again obtained by the DenseNet161 network. In Figure 3 the comparison between all the accuracy values computed is shown. CapsNet ranks fourth out of five networks, but its accuracy is not so far from the second and third ones, which are VGG16 and ResNet50.

We computed the training and the test time using Python's *datetime* library, in order to evaluate performances in terms of time. In Table 2, the execution times of training and test phases are presented, using as major parameters 30 epochs, a step size equal to 20 and a learning rate equal to 1e-3 for the traditional CNNs, while CapsNet runs with 30 epochs, a learning rate equal to 1e-2.

As described in 2, CapsNet requires a time to train that goes from 4 to 25 times the time to train of the other CNNs. Instead, CapsNet is the fastest one network in the test phase, from 4 to 8 times faster than the other CNNs. A clearer

As it can be seen from Figure 4, the CapsNet is very slow in performing the training phase, but it's very fast during the test phase, as shown in Figure 5: this can be useful in situations in which a trained model is already available and a very fast classification of new samples is required.

For the sake of completeness, we computed also the time of execution of training and test phase for the CNNs. Those values can be found in Table 3. As before, those models have been trained for 30 epochs, with a step size equal to 20 and a learning rate equal to 1e-3. In order to perform transfer learning, the *pretrained* attribute of the *Torchvision*

Network	Training time	Test time	Total time
CapsNet	NA	NA	NA
VGG16	0:34:09	0:00:07	0:34:16
AlexNet	0:07:33	0:00:05	0:07:38
DenseNet161	0:42:15	0:00:08	0:42:23
ResNet50	0:19:40	0:00:05	0:19:45

Table 3. Training and testing time for pretrained networks (h:mm:ss). Pretrained CapsNet model is not available.

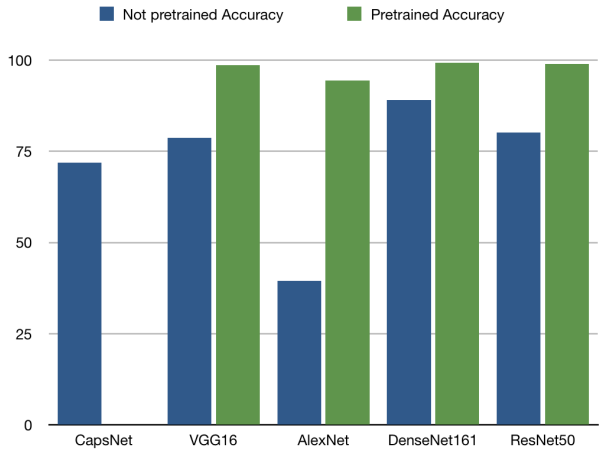

Figure 3. Comparison among not pretrained and pretrained networks' accuracies.

models has been set to True.

Capsnet has all the makings. Considering that it has not been pretrained, it reaches accuracy of 72% percent. It is really computational intensive. For this reason, performing fine tuning of hyperparameters is possible only with more efficient hardware resources. Fine tuning of hyperparameters could lead to find the best hyperparameters values for the network and to achieve higher levels of accuracy.

6. Considerations and conclusions

In this paper we wanted to evaluate the performance of a new Artificial Neural Network called CapsNet with respect to other Convolutional Neural Networks. We trained all those network on the Google Landmarks dataset, provided for the Google Landmark Recognition Challenge.

At the current state-of-art, CapsNet has not been applied to this dataset.

We can state that the results obtained with CapsNet are relevant and comparable to some of the other CNNs used, such as AlexNet. As expected, the accuracy obtained with CapsNet is not comparable to networks such as DenseNet, for example. It's necessary to notice also that CNNs such as DenseNet161 and ResNet50 are architecturally more complex than the CapsNet: they contain much more layers and connections than CapsNet.

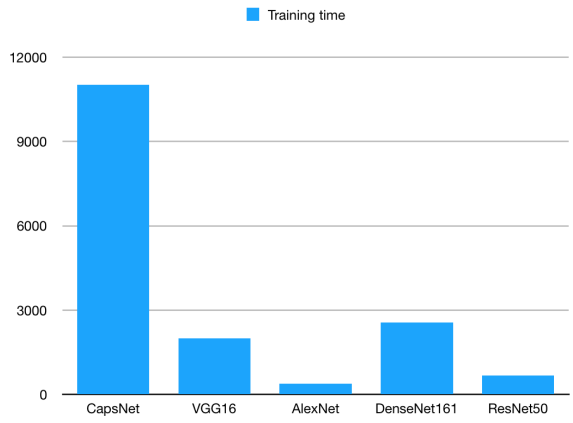

Figure 4. Comparison among not pretrained networks' training times.

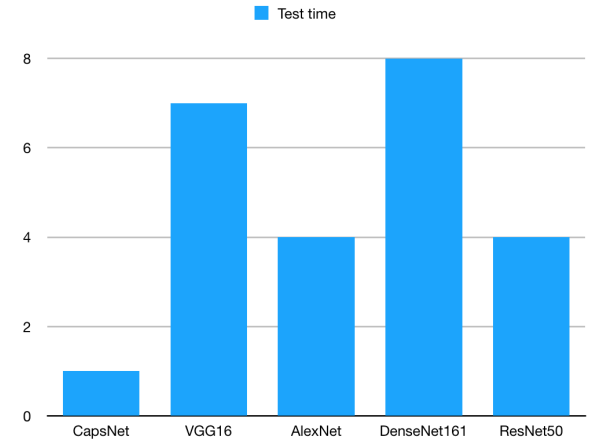

Figure 5. Comparison among not pretrained networks' test times.

We need also to take into account two limitations of CapsNet.

CapsNet has a really high computational complexity. This can be seen in Table 2. We used a relatively small dataset of about 5000 images, filtering the original Google Landmarks one. Nevertheless the CapsNet is the only network among the ones examined in which the training time exceeds 1 hour and more specifically its training time is around 3 hours. The entire Google Landmarks dataset is composed by 1225000 images, so the training CapsNet procedure could take multiple days (roughly a month using Google Colab).

The second limitation of CapsNet is that it has not been trained on ImageNet yet. Having the possibility of performing transfer learning with the CapsNet network could be a great opportunity to evaluate and compare its performance with respect to the other CNNs, such as VGG, AlexNet, DenseNet and ResNet. All of those CNNs have already been pretrained on ImageNet so transfer learning is possible. Our hypothesis is that pretraining CapsNet on ImageNet could improve its performance such that the accuracy obtained could be comparable to the one of the other CNNs, with a value greater than 94%. We are setting this threshold since CapsNet accuracy is between AlexNet and VGG16 accuracies, without transfer learning and their accuracy exploiting transfer learning reached values higher than 94%.

We can also take into account that the testing time of the CapsNet is the best with respect to all the other CNNs. This could be useful for situations in which we have an already trained network and we want to test new samples as fast as possible.

So, there is still a lot of work that could be done in future to deepen the potentialities of this new kind of network.

Firstly, it would be interesting to perform hyperparameter tuning on CapsNet to find the best performing parameters. For example, we executed the network manually with learning rate values equal to 1e-2, 1e-3 and 5e-4 and with batch size of 16 and 32. We found out that the best accuracy is obtained with a learning rate of 1e-2 and a batch size of 32. We are aware of the fact that there could be other values which obtain better results, but we did not have enough time to try all possible combinations of values.

It could be interesting to evaluate if using images of 224x224 instead of 28x28 could result into an improvement of the CapsNet performance. This was not possible to manage for us since dealing with such dimension of images saturates the quantity of RAM available on Google Colab.

After finding the best performing parameters and evaluating if the bigger images should be used as input to the network, an ultimate step could be performed. This step consists of training the CapsNet network on a larger portion of the original dataset, having more distinct classes and more training images. This could be done exploiting more powerful hardware resources.

Furthermore, as written before, it could be important to be able to train the CapsNet network on ImageNet. This would take a huge amount of spatial and temporal resources since ImageNet has over 14 million images belonging to 1000 classes. This could lead to significant improvement of CapsNet performance. Moreover, computing the top-1 and top-5 error rates on ImageNet dataset could be useful to compare CapsNet performance to CNNs performances at the state-of-art.

In the end, we argue that CapsNet could be an interesting new technique for classification of images and it leaves room to further experiments exploiting more efficient resources and modifying CapsNet architectures in order to be better suitable for the images recognition problem.

References

[1] Capsnet: Origin, characteristics, and advantages. https://missinglink.ai/guides/convolutional-neural-networks/capsnet-origin-characteristics-advantages/.

[2] G. E. Hinton A. Krizhevsky, I. Sutskever. Imagenet classification with deep convolutional neural networks, 2012.

[3] Python version of capsnet. https://github.com/dragen1860/CapsNet-Pytorch.

[4] Adit Deshpande. The 9 deep learning papers you need to know about (understanding cnns part 3, 2018.

[5] Landmark recognition challenge image downloader. https://www.kaggle.com/tobwey/landmark-recognition-challenge-image-downloader.

[6] Recognize landmarks using convolutional neural networks. https://link.medium.com/GeqNDvmxE2.

[7] Public github repository with notebooks used. https://github.com/davidetadz/landmark-recognition.

[8] *Large-Scale Image Retrieval with Attentive Deep Local Features*. Proc. ICCV'17, 2017.

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[10] S. Ren J. Sun K. He, X. Zhang. Deep residual learning for image recognition, 2015.

[11] A. Zisserman K. Simonyan. Very deep convolutional networks for large-scale image recognition, 2015.

[12] Kaggle: Google landmark recognition challenge. https://www.kaggle.com/c/landmark-retrieval-challenge/data.

[13] J. Carrillo R. Mukhometzianov. Capsnet comparative performance evaluation for image classification. University of Waterloo, ON, Canada.

[14] G. E. Hinton S. Sabour, N. Frosst. Dynamic routing between capsules. In *Advances in neural information processing systems*, 2017.

[15] G. E. Hinton S. Sabour, N. Frosst. Dynamic routing between capsules. Google Brain, 2017.

[16] Christopher J.C. Burges Yann LeCun, Corinna Cortes. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.