Plankton classification using convolutional networks

Jorge Orbegozo and Arnaud Le Doeuff

Abstract

In this document we report the application of a convolutional neural network to classify plankton's species. We have implemented the classification process using convolutional networks with keras library. We have learned the classifiers using 2/3 of the train and validate with the remaining 1/3 of the data.

Contents

1 Description of the problem

1.1 Problem

Convolutional neural networks [1] [2] [3] are ones of the most employed DNN architectures. They are particularly efficient for computer vision tasks such as image classification [4] [5].

Plankton comprises diverse small microscopic organisms such as protozoans and small crustaceans the that are the basis of the food chain in the sea. Classifying the different species that form plankton from images is a difficult problem.

The goal of the project is to design a convolutional network that outputs the probability that a given image belongs to one of the possible microrganisms that forms the plankton. The dataset was used for one of the Kaggle challenges. `https://www.kaggle.com/c/datasciencebowl/data`

1.2 Dataset

The data provided by Kaggle for the competition consisted of labeled training images and unlabeled test images which were in grayscale although they were stored in 3-channel JPEG images. Since the images were acquired from an underwater towed imaging system, they were segmented by the competition hosts such that each image contained just one species of plankton. As a result, the images were distributed unevenly among 121 classes and were of different dimensions ranging from as little as 40 pixels to as much as 400 pixels in one dimension. The exact number of images for training and test are listed here.

Training/Labeled	30,336 images
Test/Unlabeled	130,400 images

It is visible that the training set is substantially smaller than the test set and thus, in the next section we describe data augmentation to inflate the training set size.

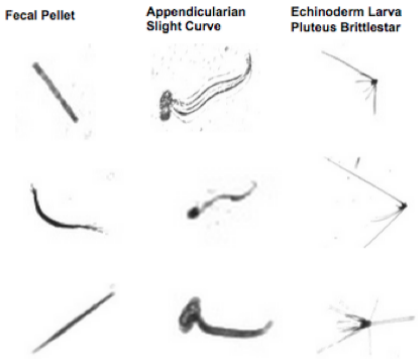


Figure 1: Plankton varieties

Some examples of labeled plankton images are shows in figure 1.

2 Description of our approach

A challenge was the size of the dataset: about 30000 examples for 121 classes. Several classes had fewer than 20 examples in total. The dataset is unbalanced. Deep learning approaches are often said to require enormous amounts of data to work well, but recently this notion has been challenged, and our results also indicate that this is not necessarily true. Judicious use of techniques to prevent overfitting such as dropout, weight decay, data augmentation, pre-training, pseudo-labeling and parameter sharing, has enabled us to train very large models with up to 27 million parameters on this

dataset.
So our approach can be resumed:

- Preprocessing the images
- Data augmentation
- Model implementation
- Training
- Validation

2.1 Preprocessing

The dataset has images of different sizes, so we have to resize them. We choose the size 128x128, because most of the images are around this value.Then we rescale the pixels to the scale 1/255. By doing this we get all the pixels to be in the gray scale. We need that, because the images of the dataset are black and white.

The data was split in to different set: train set and validation set. The train set will be use to train the model and the validation set to validate the results obtained after training.

2.2 Data augmentation

We augmented the data to artificially increase the size of the dataset. We used various affine transforms, and gradually increased the intensity of the augmentation as our models started to overfit more. We ended up with some pretty extreme augmentation parameters:

- rotation: angle of 40° (uniform)
- width translation: shift of 0.2 (uniform)
- height translation: shift of 0.2 (uniform)
- horizontal flip
- shearing: angle of 20° (uniform)
- zoom: 0.2

2.3 Model implementation

In order to do the model implementation we have to design a convolutional neural network (CNN) architecture. The best architecture of a CNN is an open topic, nowadays nobody knows exactly what designs are better to use, so we try different strategies.

The first strategy is to use few convolutional layers with a small amount of filters, one dropout layer of 0.25 and in the last fully connected layers we use only one dense layer before the one with softmax. The layers are as follow:

- Convolutional layer of 64 filters and kernel of (5,5)
- Pooling layer of (2,2) pool size
- Convolutional layer of 32 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Dropout of 0.25
- Flatten
- Dense of 256 output space
- Softmax

The second strategy is to use few convolutional layers with a small amount of filters, two dropout layer and in the last fully connected layers we try to reduce the output space and then make it bigger. The layers are as follow:

- Convolutional layer of 64 filters and kernel of (5,5)
- Pooling layer of (2,2) pool size
- Convolutional layer of 32 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Dropout of 0.25
- Convolutional layer of 16 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Dropout of 0.5
- Flatten
- Dense of 128 output space
- Dense of 50 output space
- Softmax

The third and last strategy is to use lot of convolutional layers with a big amount of filters, none dropout layers and in the last fully connected layers we use only one big dense layer before the one with softmax. The layers are as follow:

- Convolutional layer of 32 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Convolutional layer of 64 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Convolutional layer of 128 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Convolutional layer of 128 filters and kernel of (3,3)
- Pooling layer of (2,2) pool size
- Flatten
- Dense of 512 output space
- Softmax

To make it clear, the activation function in the convolutional and in the fully connected layer is relu. The loss function is the categorical crossentropy, the optimizer is adam and the metric is the accuracy. In the last layer, the activation function is the softmax with an output space of 121, one for each class.

2.4 Training

In the training phase, we use the train dataset to fit the model and compute the classifier accuracy.

2.5 Validation

To validate our results we compute the classifier accuracy in the validation dataset. This is done once the train is already finish.

3 Implementation

All the project steps were implemented in Python. We used keras to prepocess the images, do the data augmentation, divide the data in two subsets (train and validation set), make convolutional neural network architectures, train and validate the models.

To do all this things we use functions from keras, the functions are as follows:

- preprocessing.image.ImageDataGenerator

- train_datagen.flow_from_directory
- model.Sequential
- model.add
- model.compile
- model.summary
- model.fit_generator

In the following url are very good explained some of this function, this url help us a lot to understand how the functions works. `https://keras.io/api/preprocessing/image/`

Something we think needs to be clarify, is that the three models are trained and validated with the same train-set and validation-set. This is one important thing, because if the sets were done again, maybe, as they create some random data augmentation and the split is also random, some model can obtain better results because a bit of lucky data distribution. So, to make it fair, they use the same train and validation set.

We illustrate how the implementation works in out Python notebook.

4 Results

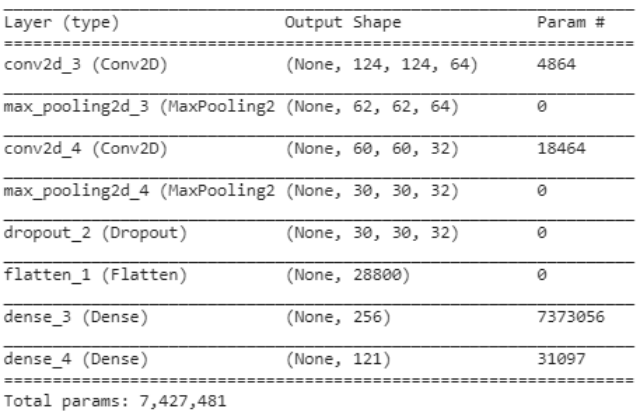
We obtain for each network the loss of the train, the accuracy of the train, the loss of the validation and the accuracy of the validation. Also, for each network we have a summary of the model so we can see how the layers are divided more easy and we can see the number of parameters that are used in every layer and also in total.

The next figures shows this tables in detail:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 124, 124, 64)	4864
max_pooling2d (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_1 (Conv2D)	(None, 60, 60, 32)	18464
max_pooling2d_1 (MaxPooling2	(None, 30, 30, 32)	0
dropout (Dropout)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 16)	4624
max_pooling2d_2 (MaxPooling2	(None, 14, 14, 16)	0
dropout_1 (Dropout)	(None, 14, 14, 16)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401536
dense_1 (Dense)	(None, 50)	6450
dense_2 (Dense)	(None, 121)	6171

Total params: 442,109

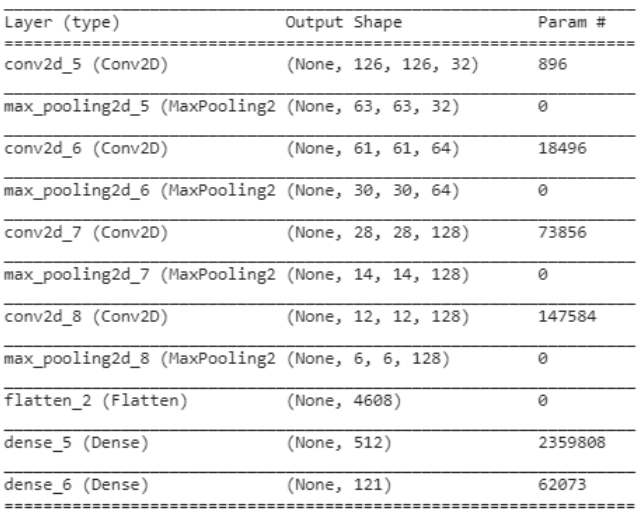
Figure 2: Network 1



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 124, 124, 64)	4864
max_pooling2d_3 (MaxPooling2	(None, 62, 62, 64)	0
conv2d_4 (Conv2D)	(None, 60, 60, 32)	18464
max_pooling2d_4 (MaxPooling2	(None, 30, 30, 32)	0
dropout_2 (Dropout)	(None, 30, 30, 32)	0
flatten_1 (Flatten)	(None, 28800)	0
dense_3 (Dense)	(None, 256)	7373056
dense_4 (Dense)	(None, 121)	31097

Total params: 7,427,481

Figure 3: Network 2



Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_5 (MaxPooling2	(None, 63, 63, 32)	0
conv2d_6 (Conv2D)	(None, 61, 61, 64)	18496
max_pooling2d_6 (MaxPooling2	(None, 30, 30, 64)	0
conv2d_7 (Conv2D)	(None, 28, 28, 128)	73856
max_pooling2d_7 (MaxPooling2	(None, 14, 14, 128)	0
conv2d_8 (Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_8 (MaxPooling2	(None, 6, 6, 128)	0
flatten_2 (Flatten)	(None, 4608)	0
dense_5 (Dense)	(None, 512)	2359808
dense_6 (Dense)	(None, 121)	62073

Total params: 2,662,713

Figure 4: Network 3

The results are summarizes in the following table:

Nº network	Train loss	Accuracy train	Validation loss	Accuracy validation
Network 1	3.0447	24.49%	3.1295	23.90%
Network 2	3.0600	25.02%	3.0385	24.60%
Network 3	2.3517	36.63%	2.3739	34.90%

5 Conclusion

To conclude we tried different strategy to predict the species of plankton. The best architecture is the number three.

As can be seen in the previous table, the networks 1 and 2 have similar accuracy. Our conclusion about this is that, since both use dropout and the third network not, useful information is lost by applying dropout. Also, in the convolutional layers, in the first and second models, we are decreasing the number of filter for every new layer. Starting in 64 filters and decreasing them to 32 in the first model and to 16 in the second one. This can be another reason to have less accuracy than in the third model.

Another reason to get more accuracy can be the number of layers. We start by putting very few layers, but when we decide to increase the number of layers with a lot of convolutional layers and a lot of pooling layers, we get better results.

So as final conclusion:

- more layers get better accuracy
- dropout lose a lot of relevant information
- increasing the number of filters in convolutional layers is better than decreasing them

References

[1] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high perfomance convolutional neural networks for image classification. *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, 22:1237, Barcelona, Spain, 2011.

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Concolutional architecture for fast feature embedding. *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, ACM, 2014.

[3] Yoom Kim. Convolutional neural networks for sentence classification. *CoRR*, arXiv:1408.5882, 2014.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.