Image classification with Neural Networks

Picascia Sergio
Data Science and Economics
University of Milan

Abstract

Image classification is one of the greatest domains in which neural networks play an important role. The applications of this task are infinite, spacing from the more casual ones, like automated image organisation, to marketing and medicine. It received a lot of attention in the machine learning field during the last years thanks to the use of deep learning; in particular convolutional neural networks, CNNs or ConvNets, perform extremely well on classifying images. I will explain why in this paper and I will try to build an image classifier based on the dataset Fruit 360, a collection of pictures of fruits and vegetables. Different networks with distinct architecture are trained in order to reach the best possible performances in terms of validation accuracy and loss. We will see how VGG-blocks achieve this goal and which regularisation techniques it is possible to implement in order to improve the networks.

Introduction

The objective of this project is to build an image classifier with neural networks. The dataset is available on a Kaggle repository and it is called Fruit 360[1]: it consists in images of different varieties of fruits and vegetables, taken from a short movie recorded on white background. The original dataset contained 131 distinct classes and it was already split between training and test set; for simplicity, in this analysis I considered only the 10 most frequent type and assigned each variety to the correspondent label among the following: apple, banana, cherry, grape, peach, pear, pepper, plum, potato, tomato. The resulting dataset is composed of 34528 training images and 11551 test images. Each picture is of size 100x100 and is in JPG format, so before starting the research I had to scale them down and convert to RGB channels. Thanks to the great performances that convolutional neural networks have on image classification, I used them instead of simple feed forward networks. I performed the analysis applying different architectures in order to understand which one would perform better.

[1]Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42, 2018.

The paper introduces the dataset and an implementation of a Neural Network trained to recognized the fruits in the dataset.

Figure 1: Sample of images from the dataset.

Analysis

Image classification is a supervised learning problem and it is one of the greatest field of application for deep neural networks. It has been shown that convolutional neural networks perform extremely well in this task, while basic feed forward neural networks struggles, due to their complexity and the need of estimating a large number of parameters. Furthermore, when analysing images, reshaping the matrix of pixel into a vector let us loose a lot of spatial interaction between the pixels; with CNNs, instead, adjacency information is preserved using convolution layers.

In these type of layers, a filter or kernel shifts along the image performing matrix multiplication; the speed at which it moves is indicated by the stride length. The convolution layer captures high-level features from the picture and it is possible to stack more of them in order to extract more and more features. The extracted features can have the same size of the original picture, in case of same padding, or reduced size, valid padding; in the first case we add a line of pixel with value zero around the image, in order to momentarily increase the size of the picture.

Another type of layer that is commonly used in combination with the convolutional one, is the pooling layer: it reduces the spatial size too and it is useful for extracting dominant features. The most used one is max pooling, which returns the maximum value from the portion of image covered by the kernel, but it is also possible to use the average pooling that, as the name says, computes the average of the values.

A CNNs usually ends with a fully connected layer that then passes the information to the output layer, where it assigns the predicted label using the softmax function.

During the learning process, instead, it is preferable to use the Rectified Linear activation function (ReL), which returns 0 if the input is less or equal to 0, otherwise it returns the value of the input itself. It is chosen above the others like the sigmoid or the hyperbolic tangent, for two main reasons: it does not saturate, because for extreme values it doesn't get stuck, and it can deal with the problem of vanishing gradients that prevents the network to learn properly. Moreover, it is easy to compute and, even though it is nonlinear, it almost behaves as a linear function. I am going to use it in my project because, thanks to the advantages previously discussed, it has recently become the standard ac-

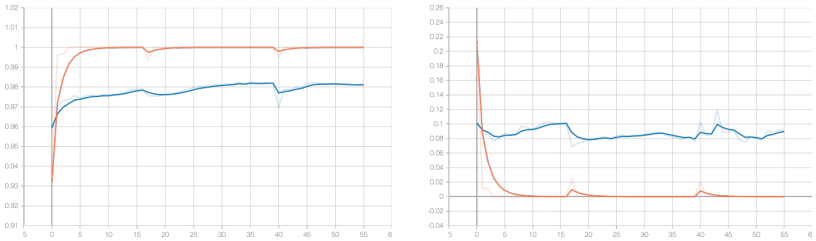

Figure 2: One VGG-block: accuracy (left) and loss (right).

```
Epoch 36/100
1078/1079 [============================>.] - ETA: 0s - loss: 7.2847e-09 - accuracy: 1.0000
Epoch 00036: val_accuracy improved from 0.98234 to 0.98260, saving model to vgg1.h5
1079/1079 [==============================] - 69s 64ms/step - loss: 7.2952e-09 - accuracy: 1.0000 - val_loss: 0.0822 - val_accuracy: 0.9826
```

Figure 3: One VGG-block: results.

tivation function for CNNs.

As optimiser, I chose Adam which is preferred over other stochastic methods when running deep neural networks. In fact, it combines advantages from two stochastic gradient descent expansions, AdaGrad and RMSProp, adapting its parameter learning rates using second moments and improving performances on sparse gradients and noisy problems.

Finally, for the loss function, I used the categorical cross-entropy loss for multi-class classification tasks, which is a very good measure of distinctness between two probability distributions.

For all the models, I used early stopping, a regularisation technique that prevents overfitting concluding the learning process when there are no significant improvements on the test set for 20 epochs in a row.

VGG-blocks

The first architecture I am going to use is the VGG network. It has shown great performances on image classification and it is one of the best structure to apply. It is based on VGG-blocks, which are composed by convolutional layers with small filters, usually 3x3, same padding and a nonlinear activation function (ReLU), followed by a max pooling layer of size 2x2 with strides 2.

After the first part composed by convolutional blocks, the network has a second part of fully connected layers. The sum of convolutional layers and fully connected one, gives the name to VGG network: for example, since I built three VGG networks with respectively one (Figure 2 and 3), two (Figure 4 and 5) and three (Figure 6 and 7) convolutional blocks, I ended up training a VGG-4, VGG-6 and VGG-8. Best performances have been found on VGG-16 and VGG-19 but, given the simplicity of the task, I decided to stick with more basic networks.

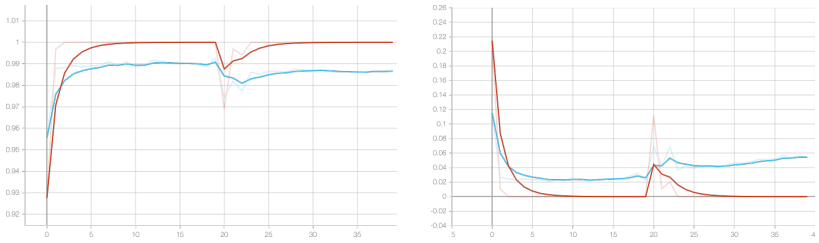

Figure 4: Two VGG-block: accuracy (left) and loss (right).

```
Epoch 20/100
1078/1079 [============================>.] - ETA: 0s - loss: 2.7128e-09 - accuracy: 1.0000
Epoch 00020: val_accuracy improved from 0.99178 to 0.99273, saving model to vgg2.h5
1079/1079 [==============================] - 74s 69ms/step - loss: 2.7102e-09 - accuracy: 1.0000 - val_loss: 0.0219 - val_accuracy: 0.9927
```

Figure 5: Two VGG-block: results.

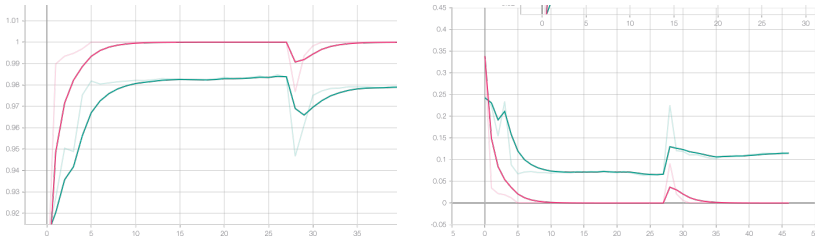

Figure 6: Three VGG-block: accuracy (left) and loss (right).

```
Epoch 27/100
1078/1079 [============================>.] - ETA: 0s - loss: 4.0087e-10 - accuracy: 1.0000
Epoch 00027: val_accuracy improved from 0.98416 to 0.98476, saving model to vgg3.h5
1079/1079 [==============================] - 79s 73ms/step - loss: 4.0049e-10 - accuracy: 1.0000 - val_loss: 0.0646 - val_accuracy: 0.9848
```

Figure 7: Three VGG-block: results.

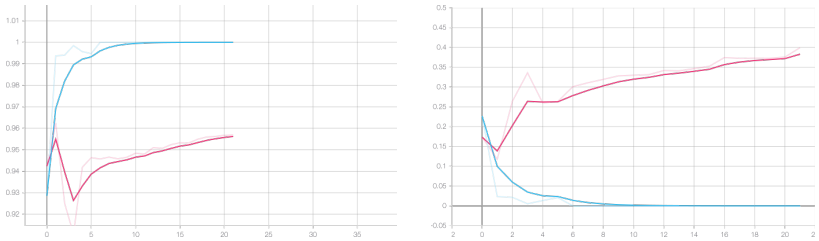

Figure 8: ResNet: accuracy (left) and loss (right).

```
Epoch 2/100
1077/1079 [============================>.] - ETA: 0s - loss: 0.0234 - accuracy: 0.9937
Epoch 00002: val_accuracy improved from 0.94234 to 0.96251, saving model to /content/drive/My Drive/ML Project/resnet.h5
1079/1079 [==============================] - 47s 43ms/step - loss: 0.0234 - accuracy: 0.9937 - val_loss: 0.1177 - val_accuracy: 0.9625
```

Figure 9: ResNet: results.

ResNet

ResNet is a CNN architecture that received a lot of attention thanks to the innovative approach to the vanishing gradient problem, discussed above. The solution is given by the residual module, which is a block of convolutional layers with the same number of small filters 3x3 and a shortcut connection that adds the inputs of the block to the outputs of the last layer in the block (Figure 8 and 9).

Inception

The Inception architecture, also known as GoogLeNet, is based on the idea of having a large number of convolutional layers in order to go deeper in the network without incurring in problems like overfitting or high computational resources. It consists in parallel convolutional layers with different sized filters, followed by a max pooling layer and then concatenated together. I first train a naive version (Figure 10 and 11) of the Inception algorithm and then, in the second version (Figure 12 and 13), I add a convolutional layers 1x1 before all the other convolutional layers in order to reduce the computational resources required.

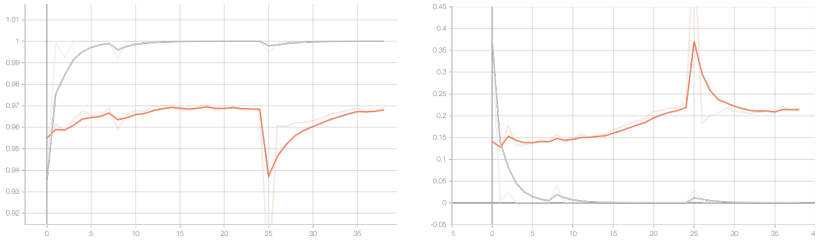

Figure 10: Inception (naive): accuracy (left) and loss (right).

```
Epoch 19/100
1079/1079 [==============================] - ETA: 0s - loss: 9.7341e-07 - accuracy: 1.0000
Epoch 00019: val_accuracy improved from 0.97005 to 0.97039, saving model to inc_naive.h5
1079/1079 [==============================] - 96s 89ms/step - loss: 9.7341e-07 - accuracy: 1.0000 - val_loss: 0.1889 - val_accuracy: 0.9704
```

Figure 11: Inception (naive): results.

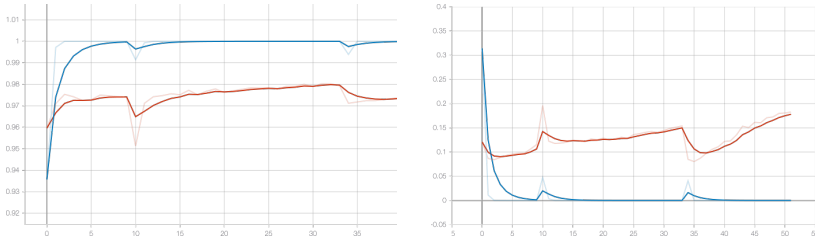

Figure 12: Inception with dimensionality reduction: accuracy (left) and loss (right).

```
Epoch 32/100
1079/1079 [==============================] - ETA: 0s - loss: 4.0671e-09 - accuracy: 1.0000
Epoch 00032: val_accuracy improved from 0.98000 to 0.98026, saving model to inc_dr.h5
1079/1079 [==============================] - 143s 133ms/step - loss: 4.0671e-09 - accuracy: 1.0000 - val_loss: 0.1488 - val_accuracy: 0.9803
```

Figure 13: Inception with dimensionality reduction: results.

Final details

Since the best model so far has been the network with two VGG-blocks, I tried to implement a few improvements to it. First of all, since I am using the ReLU activation function, I decided to use the He initialisation for kernels, which has been demonstrated to perform better in this context.

Then, I added batch normalisation layers: it is a technique that normalised the inputs, so it rescales them to standard Gaussian, via adjusting the activation function. This improves the performances, the stability and the speed of the network. Thus, I eliminated the bias from the layers, since batch normalisation already introduce the variables gamma and beta (Figure 14 and 15).

Another regularisation method is the dropout, which is useful in order to prevent overfitting. It adds some noise to the learning problem leaving out some of the neurons of a layer. I added dropout layers after both convolutional blocks and fully connected layer, with a rate of 0.2 (Figure 16 and 17).

Finally, I modified the input images with data augmentation, a method which is very helpful when we do not have at disposal a large amount of data. It allows to transform the pictures by flipping, cropping, shifting and rotating them, or by zooming in or out or changing their brightness (Figure 18 and 19).

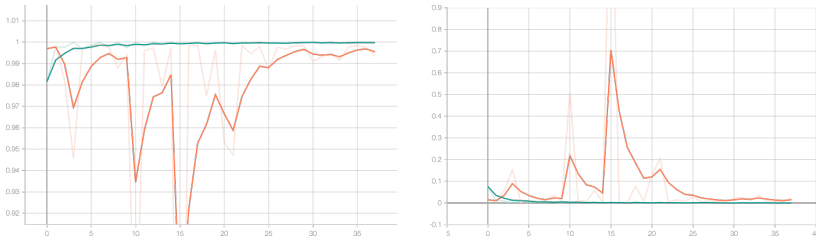

Figure 14: Two VGG-block with Batch Normalization: accuracy (left) and loss (right).

```
Epoch 18/100
1077/1079 [============================>.] - ETA: 0s - loss: 6.4913e-05 - accuracy: 1.0000
Epoch 00018: val_accuracy improved from 0.99870 to 0.99879, saving model to /content/drive/My Drive/ML Project/vgg2_bn.h5
1079/1079 [==============================] - 49s 46ms/step - loss: 6.4797e-05 - accuracy: 1.0000 - val_loss: 0.0031 - val_accuracy: 0.9988
```

Figure 15: Two VGG-block with Batch Normalization: results.

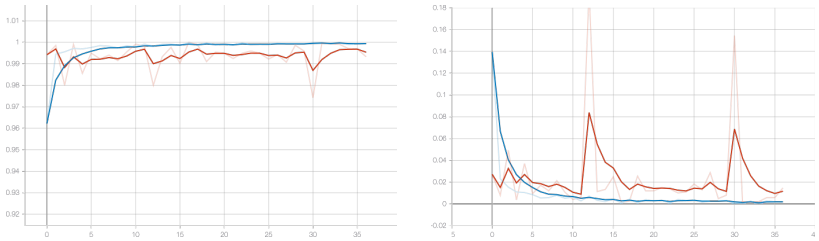

Figure 16: Two VGG-block with Batch Normalization and Dropout: accuracy (left) and loss (right).

```
Epoch 17/100
1078/1079 [============================>.] - ETA: 0s - loss: 9.6485e-04 - accuracy: 0.9998
Epoch 00017: val_accuracy improved from 0.99913 to 1.00000, saving model to /content/drive/My Drive/ML Project/vgg2_bn_d.h5
1079/1079 [==============================] - 50s 47ms/step - loss: 9.6396e-04 - accuracy: 0.9998 - val_loss: 0.0013 - val_accuracy: 1.0000
```

Figure 17: Two VGG-block with Batch Normalization and Dropout: results.

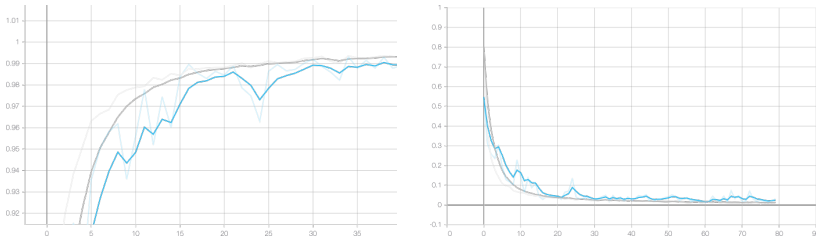

Figure 18: Two VGG-block with Batch Normalization, Dropout and Data Augmentation: accuracy (left) and loss (right).

```
Epoch 60/100
1078/1079 [============================>.] - ETA: 0s - loss: 0.0119 - accuracy: 0.9957
Epoch 00060: val_accuracy improved from 0.99446 to 0.99558, saving model to /content/drive/My Drive/ML Project/vgg2_aug.h5
1079/1079 [==============================] - 78s 72ms/step - loss: 0.0119 - accuracy: 0.9957 - val_loss: 0.0140 - val_accuracy: 0.9956
```

Figure 19: Two VGG-block with Batch Normalization, Dropout and Data Augmentation: results.

Conclusion

Among all the models, the one that reached the highest performances is the network with two VGG-blocks, batch normalisation and dropout: it was able to achieve a test accuracy of 1 and a loss of 0.0013. Both the Inception (0.9803) and ResNet (0.9625) networks performed even worse than the one VGG-block (0.9826), while the network with three VGG-blocks reached a validation accuracy of 0.9848, slightly behind the basic model with two VGG-blocks (0.9927). Adding more layers can sometimes worsen the model rather than improving it, and that was the case. Applying data augmentation to the inputs, surely challenged the network, that reached a train accuracy of 0.9956. Overall, all the scores are pretty high, showing that the CNNs are able to properly tackle the assigned task. From there, it could be possible to apply the same architecture to the whole dataset, with all the 131 classes, in order to test at which extend these models are generalisable.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.