UNIVERSITÀ DEGLI STUDI DI MILANO

Università degli Studi di Milano

Algorithm for Massive Data

Face – comic recognizer using convolutional neural network

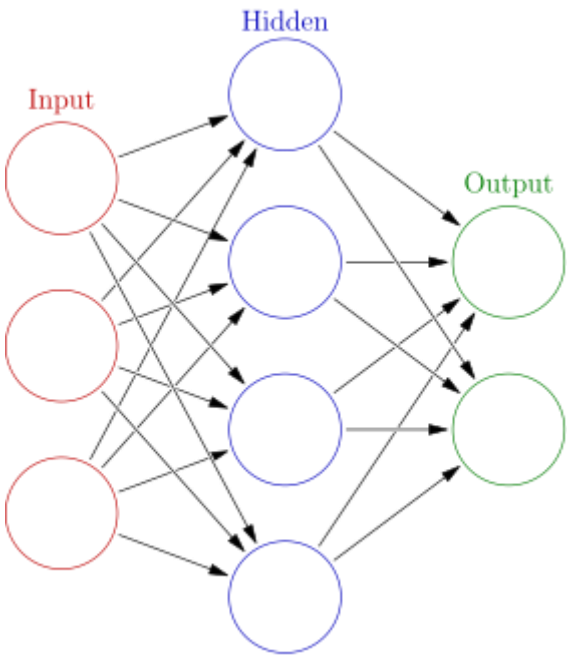
Stefano De Filippis

Abstract

Convolutional neural networks are algorithm very useful in classifying images and their architecture is inspired from the mechanism behind the human brain, in particular how it processes images. The aim of this work is to build a convolutional neural network able to recognize a human face to his comic version.

INTRODUCTION

Neural networks are algorithms which take inspiration from the structure of the human brain. Of course, the human brain is the most complex machine in the world and we cannot talk about an approximation, but the structure of these kind of algorithms are very similar to it. A neural network is composed by nodes (the neurons) creating a layer, which are connected to nodes in another layer with some edges which could be seen as synapses. However, in this case the edges are represented by weights which multiply the values inside the nodes. At the end of this network there is an output layer which gives the result. An example of a neural network's architecture is the following:



We can notice that there is an input layer, which is the one representing the training observations. Here each node is attached to a feature's value measured in the observation. So, if the observations are represented with 50 dimensions, the input layer will have got 50 nodes. Then there is a hidden layer whose number of nodes must be decided, so it is an hyperparameter. In this layer, each node is connected with each input coming from the previous layer and for this reason the layer is called "fully connected" or "dense". At the end of each neural network there is an output layer giving the results, in this case it is composed by two nodes. There could be more than one hidden layer and also the number of them is an hyperparameter.

A neural network could make classification and regression tasks and the number of nodes in the output layer changes. For example, in a regression context the output layer would have, in general, one node representing the value of the prediction, like a stock price. Instead in a classification context the output node would have a number of nodes equal to the number of classes.

As said before, the edges in the graph are weights which multiply values inside nodes and they produce an output, which is called "logit":

$$logit = [x_1 \quad x_2 \quad \dots \quad x_n] \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} - b$$

Where the x_i is the input coming from the previous layer, the term w_i represents the weight and b is a number called bias.

However, the final output in a node is not the logit but is the result of a function whose argument is the logit, these functions are called "activation functions".

There are different types of activation functions, one of the most known are: ReLU, Sigmoid and Softmax. In general, all the activation functions must satisfy properties like being continuous and differentiable everywhere (or almost everywhere), the derivative must not become very small in the range values of the inputs and at the same time it must not become too large for a matter of computation stability.

The ReLU is the following:

$$f(x) = \max(0, x)$$

Instead, the Sigmoid, whose name derives from the characteristic "S" shape, is described in this way:

$$f(x) = \frac{e^{\beta x}}{1 + e^{\beta x}}$$

Where the β changes the steepness of the function.

The Softmax is the following:

$$f(x_i) = \frac{e^{x_i}}{\sum_j^n e^{x_j}}$$

Where n is the number of outputs. This last function is different from the Sigmoid because it is does not operate elementwise in the vector.

The last two functions take values between zero and one, for this reason they could be interpreted as probability distribution. Consecutively they could be used as activation functions of the final layer (the output one) of the neural network in a classification problem context, giving the probability that an observation belongs to the classes. However, the Softmax tends to push only one value to one, in this way the uncertainty regarding the classification is reduced with respect to the Sigmoid function.

At the end, it is necessary a function also for the loss, in a regression context could be useful to compute the squared loss $L(y, y') = (y - y')^2$. In a classification task, it is more used the categorical cross entropy loss, which is the following:

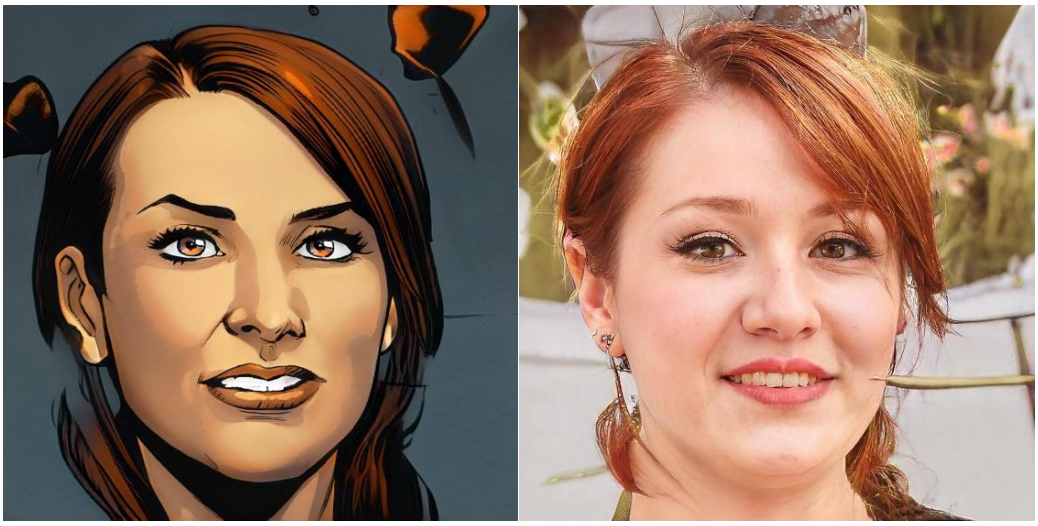
$$L(y - y') = \sum_{i=1}^{k} y_i \cdot \log\left(y'_i\right)$$

Where k is the number of classes.

DATA AND METHODS

Data:

In this work the aim is to build a neural network able to recognize a real face against his comic version. The dataset contains 20000 observations. In this case the classes are two: real face or comic. These two classes are evenly separated, so there are 10000 observations for each of them. This is an example of images:

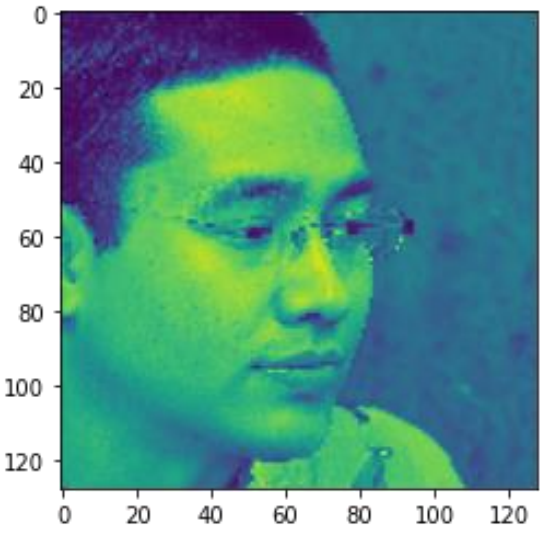


Since this is a classification problem related to images, it is necessary using a "convolutional neural network" rather than a classical neural network like the one shown in the introduction. These models work with tensors which are matrices in three dimensions, so in addition to the width and length, a tensor has got also the depth. A color image could be seen as a tensor with a depth equal to three, so it is a three-dimensional matrix composed by three matrices in two dimensions. The first matrix has got pixel values for the color red, the second one for color green and the third one for color blue. So, the input in this case has got a very huge size.

In this work, the images used are in 1024x1024 format and this means that a first input layer would have $1024 \cdot 1024 \cdot 3 = 3145728$ nodes, which is an enormous amount. If this input node is linked to a dense hidden layer in a classical architecture of a neural network, the number of parameters to estimate will be very large. Even a dense hidden layer with 10 nodes needs to estimate more than 30 million of parameters. With such a big amount of parameters it is very easy to overfit, at the same time estimating them is also very heavy from a computational point of view.

For this reason, convolutional neural networks are largely used in analyzing images, because they need to estimate an amount of parameters very low with respect to a classical neural network and these are the models used in this work.

As said before, a color image is represented by a tensor with a depth equal to three, however I decide to convert the images in a version with one channel. Basically, I compute the mean along the third dimension (the depth dimension) and divide it by 255 so that all the pixel values are between 0 and 1. In this way the mathematical representation of an image is the classical two-dimensional matrix. But the manipulation of the inputs does not stop here, I also decide to decrease the format, since a 1024x1024 resolution is very high, so I reshape them into a 128x128 pixel format. In this way the size of inputs is a bit decreased. An example of image after this manipulation is the following:



For the labels I use the so called “one-hot-encoding”, where I attach to each image a vector with two entries which are only equal to one or zero, according to the belonging class of the image.

At this point, I split the dataset into training and test set, where the first one contains 13332 images, instead the second one has got a number of examples equal to 6666. All these sets has got an equal amount of faces and comics images.

Stochastic gradient decent:

A neural network is trained using the gradient descent. The gradient is computed on the loss function and it represents the direction where this function increases the most, so the update of the parameters is done in the opposite direction of the gradient. The magnitude of the update is

made considering a learning rate which multiplies the negative gradient. The general mechanism of this algorithm is the following:

$$w_{t+1} = w_t - \eta \nabla_w(L)$$

Where:

- w_{t+1} is the vector of weights at time $t + 1$
- w_t is the vector of weights at time t
- η is the learning rate
- $\nabla_w(L)$ is the gradient computed on the loss L

However, computing the gradient considering all the examples in the training set is very heavy, for this reason it is used a different version of the gradient descent called "stochastic gradient descent". In this method, the gradient is computed for a number of epochs, in each epoch it is considered not all the observations but a portion of them called "batch". In this work, the batch is a random draw over the training set with a fixed size. The aim is to use a batch size not so large in order to speed up the computation of the gradient. So, at the epoch t the parameters are updated considering a batch, at the next epoch $t + 1$ the previous parameters are updated again considering another batch. This procedure is iterated for a fix number of epochs. Therefore, the number of epochs could influence the parameters and consecutively the performance of a neural network, because a bigger number of epochs means a more trained network. This is also the approach used in this work. In particular, I fix the batch size to 100, because I notice that a bigger size makes the computation far slower. At the same time, I consider a number of 100 observations inside a batch a good approximation of the training set.

Models:

First of all, I divide the dataset in two sets, training and test. All these sets are perfectly balanced with a number of observations equal to 13332 for the training one (6666 faces and 6666 comics) whereas the test one contains 6666 examples (3333 faces and 3333 comics). I create three different convolutional "architectures", where they share the number of dense layers which is equal to two, the usage of max pooling layers and a filter size equal to 3 for the convolutional layers.

In particular: the first structure contains only one convolutional layer, the second one a number equal to three and the last one contains five convolutional layers. Of course, all these layers are followed by an equal amount of max pooling layers.

As stated above, the filter has a size equal to 3 in all the models, these filters could be seen as "matrix" which examines small spatially contiguous area of the image multiplying the entries in that area for the values inside the filter. The output is the summation of these multiplication summed to a bias term, exactly like in a classical neural network. A filter could be seen as an element which tries to capture a particular feature, and since this feature could be seen in other part of the image, the filter is applied to many locations of the input. We can slide the filter for a certain number of pixels at a time, the term denoting the sliding is called "stride" and his values denotes also the number of pixels. The usage of such filters could change the shape of the output, however there is a technique calling "zero padding" that adds a row of only zeros to each side of the input matrix and it allows to maintain the same shape. In addition, using a zero padding could lead to a better analysis of the image borders. It exists a formula which allows to compute the final shape of the output on the basis of the types of parameters used:

$$n = \frac{m - f + 2p}{s} + 1$$

Where:

- m is the size of the original matrix
- f is the size of the filter
- p is the number of zeros rows and columns applied to each side of the matrix
- s is the value of the stride

A max pooling layer scans a region of the matrix and take the maximum value inside that region. This procedure is repeated in all the parts of the matrix like the "stride" for the filter. The max pooling layer has got the scope to decrease the width and height of the matrix. I decide to use a 2x2 filter in all the three structures and this allows to reduce by one half the shape of the two-dimensional matrix. At the end of each structure there are two "dense" layers, they are called like this because they are fully connected with the inputs. The last dense layer is also the output one and the number of his nodes is equal to two for all the three architecture.

Grid search:

As said before, I create three architecture which share some hyperparameters, but other ones change in order to create different models. I summarize what the final models have in common and what they do not:

- Common: filter size equal to 3, usage of max pooling layers, 2 dense layers and the ReLU activation function for all the layers except the last one which uses the Softmax. The reason of the last choice is that the ReLU activation function does not saturate so easily like the Sigmoid one and at the same time it speeds up the computation of the gradient, at the end the Softmax gives a probability distribution for the classes.
- Different: the number of convolutional layers (the first has got 1, the second 3, the third 5), the number of filters, the usage of zero padding or not and the number of nodes in the first dense layer.

The number of filters tested are:

1. First structure: [10], [32]
2. Second structure: [10, 20 ,30], [32, 64, 128]
3. Third structure: [10, 20, 30, 40 ,50], [32, 64, 128, 256, 512]

Instead, the number of nodes inside the dense layer is 10 or 20 for all the structures.

At the end, each convolutional neural network has three different numbers of convolutional layers, two possible number of filters, have or not the zero padding and a two number of nodes in the dense layer. Consequently, the number of combinations is 3x2x2x2 = 24 models.

In order to train these models, I make a random draw from the training set with replacement for 10000 times and I fix them, so that all the models are trained with the same images. These observations are divided in groups of 100 and each of these groups represent a batch. So at the end there are 100 batches with 100 observations inside. At the same time, I fix also the initializer for the parameters, I decide to use values coming from a Normal distribution with mean 0 and standard deviation equal to 0.05. I set the seed in a different way in order to not have the same starting weights, because in that case there could be some problems during the update. Every model, after the training phase, is tested in the test set.

Using images randomly drawn with replacement, the probability to have seen at least one time an observation inside the training set is equal to 52,77%, so every model is trained considering more than one half of observations inside the training set (considering all the epochs). This probability is computed in the following way:

$$1 - \left(1 - \frac{1}{13332}\right)^{10000} = 0{,}5277$$

Where 13332 is the number of observations inside the training set. The term $1 - \frac{1}{13332}$ represents the probability that an observation is not drawn and it is elevated to the power of 10000 which is the number of times the random draw is repeated.

However, in my case, the fraction of unique observations is equal to 52,99% which is even a bit more than his theoretical value. In addition, the proportion of images' classes randomly drawn is almost the same, 50,14% of faces (with 3530 unique images) and 49,86% of comics (with 3535 unique images). Consequently, all the CNN are trained with an almost equal amount of comics and faces images.

The grid search returns the best model (according to the performance inside the test set) which has:

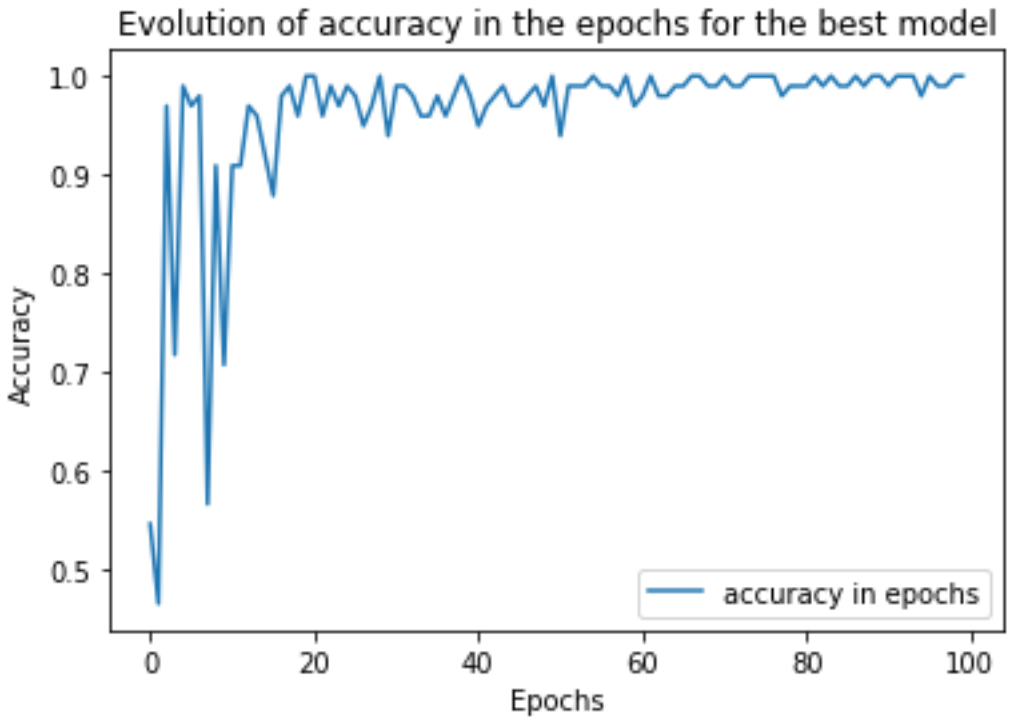
- 5 convolutional layers
- [32, 64, 128, 256, 512] filters
- No zero padding
- 10 nodes in the first dense layer

So the final structure is summarized in the next page.

Layer (type)	Output Shape	Param #
conv2d_71 (Conv2D)	multiple	320
max_pooling2d_58 (MaxPooling2D)	multiple	0
conv2d_72 (Conv2D)	multiple	18496
max_pooling2d_59 (MaxPooling2D)	multiple	0
conv2d_73 (Conv2D)	multiple	73856
max_pooling2d_60 (MaxPooling2D)	multiple	0
conv2d_74 (Conv2D)	multiple	295168
max_pooling2d_61 (MaxPooling2D)	multiple	0
conv2d_75 (Conv2D)	multiple	1180160
max_pooling2d_62 (MaxPooling2D)	multiple	0
flatten_17 (Flatten)	multiple	0
dense_24 (Dense)	multiple	20490
dense_25 (Dense)	multiple	22

Total params: 1,588,512

It has got almost 1,6 million of parameters and the graph below summarizes the evolution of accuracy in the epochs during the training.



Thanks to the graph is possible to see the evolution of the model during the training: in the first epochs the situation seems very messy, because it passes from high accuracy, for the batch examined, to a very low one. This is due to the fact that the parameters are not already updated well and each epoch analyze a batch of images potentially different from the ones analyzed in the previous epoch, since the batches are composed by a random draw with replacement from the training set. So, the CNN needs other epochs in order to achieve a better performance. After the twentieth epoch the accuracy starts to be high, almost always higher than 95% and after the fiftieth it is almost always around one.

The accuracy of the model for the only observations used during the training phase is equal to 99,69%. So, the training error is equal to 0,31% circa.

Instead, considering the observations inside the training set not included during the training (for this reason also them could be seen as “test” observations) the accuracy is about 99,70% circa.

In the test set all the observations where completely excluded by the training, so the model has never seen them before, and the accuracy here is equal to 99,65%. In particular, the accuracy for the two classes is equal to 99,73% for comics and 99,57% for faces and this is not a very large difference. However, according to only the performance inside the test set, it seems that this model has a better performance in recognizing comics, probably because these kinds of images are easier,

since they are drawings and they present less variety of nuances and lights effects instead of a picture of a real face.

Considering the level of accuracy for the training and for the observations not included in the training process, I can say that this model does not overfit.

CONCLUSION

Neural networks are algorithm which are able to reach very high accuracy and there are different types of architecture. This work has got the aim to recognize faces to their comic version and a classical neural network structure could give some problems, like overfitting, and it also could not be very accurate. For this reason, I use a convolutional neural network and I create three structures with a different number of layers. The other parameters that changes are the number of filters, the usage of zero padding or not and the number of nodes in the first dense layer after the convolutional ones. These combinations lead to an estimate of 24 models, all of them are trained with the same images randomly drawn from the training set. In particular, I repeat the random draw with replacement 10000 times and, from theoretical point of view, the probability that an image is drawn is around 52,77%. However, in my case the total number of unique images is around 52,99%. I divide these 10000 images in 100 batches of size 100, consequently the number of epochs is equal to 100 too. In each epoch, I optimize the parameter considering one batch at time. During the grid search, every model is tested in the test set and the best one is the one with 5 convolutional layers, with [32, 64, 128, 256, 512] filters, without zero padding and with 10 nodes inside the first dense layer. This model has a performance on the images with which it is trained around 99,69%, whereas in the test set its performance is about 99,65%. The performance inside the classes is different but not too much, in particular this network has less difficulties in recognizing comic faces instead real ones, with an accuracy of 99,73% in the first and 99,57% for the latter. One possible reason for this situation is that comics images has got less variety of nuances and light effects than a picture of a real face, for this reason maybe the convolutional layers in this network has learnt better how to recognize these esthetical features.

REFERENCES

- Lectures from course Algorithm for Massive Data
- *Mining of Massive Datasets*, written by A. Rajaraman e J. Ullman