Deep Learning Image Classification

Statistical Methods for Machine Learning[*]

Filippo Menegatti[†]

[*]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 offenses 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/us or any other person for assessment on this or any other course of study.

[†]Data Science and Economics, University of Milan

Contents

1 Introduction

In these essay we are going to analyze the data set available on Kaggle website [1] under the license **CC BY-SA 4.0**, using a Deep Learning approach. The data set contains 90380 images of 131 fruits and vegetables divided in folders for training and test set respectively. We are going to select just a subsample of the available fruits creating 10 macrocategories with the most frequent types. Different Neural Networks will be compared, starting from different settings, and then tuning its hyperparameters in order to measure the performance of different architectures and optimizers. We are going to conclude the essay with the application of two famous Convolutional Neural Network's architectures: the VGG-16 and the Res-Net 34.

2 Short theoretical Background

2.1 Deep Neural Networks

Neural Networks (NNs) are a specific kind of model conceived for the first time in 1943 from McColluch and Pitts [2] but it went unexplored for quite a long time due to the fact that we did not have enough computational power to exploit the most of their potential. During the 90s the interest on Neural Networks returned to raise because of the increasing quantity of data available to feed them, and the huge and continuous increase of computational power.
Common Neural Network models are defined *feedforward* because the information passes through the different layers of the model until the output, and in a sense there are - generally - no feedback connections.
The network structure can be graphically represented with a *Directed Acyclic Graph* (DAG) $G = (V, E)$ where V are the vertices (nodes) and E are the edges or connections between them. In each node a function $g(x) = \sigma(w^T x)$ - where σ is known as *Activation Function* - is applied. The layers are named differently depending on the position they belong to: the first and the last layer are respectively known as input and output layer, while the ones in the middle are named *hidden layers* because their output is not directly shown. The number of layers defines the width of the model which are also defined

deep due to the fact that they can be composed by many hidden layers.
The research concerning the best structure - or architecture - of a NN for a given set of data is nowadays a quite active area and a final answer does not exist, for this reason the process of designing the *best* model consists on a trial and error tuning process that aims at gradually increase the overall performances.

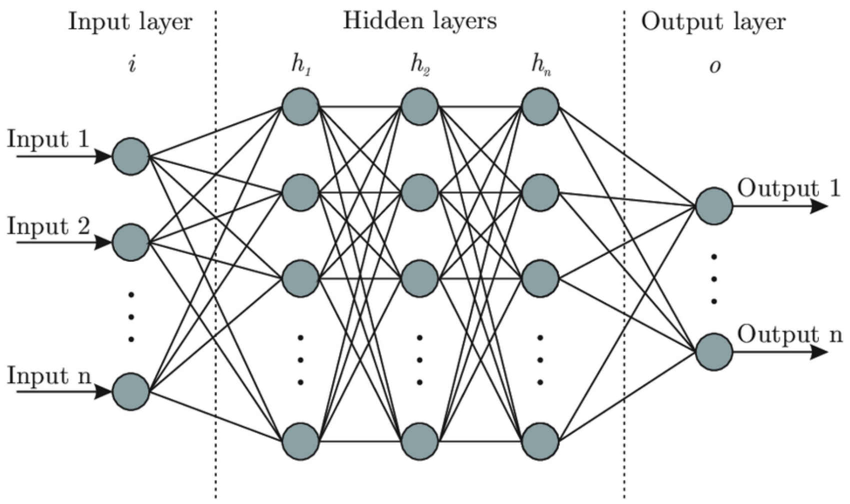


Figure 1: Deep Neural Network

2.1.1 Activation Function

The activation function we indicated earlier with $\sigma(\cdot)$ can take many different forms depending on the kind of problem we are facing. In our analysis we are going to use just two specific variants: the *Rectified Linear Units* (ReLU) and the *Softmax*.
The ReLU is a function of the kind relu(z) $= \max\{0, z\}$ which returns the input itself if it is positive and zero otherwise [3]. It is one of the most used activation functions for the layout of neural networks' hidden layers, this is due to its computational simplicity, its linear behavior and the ability - named sparse representation - of outputting *real* zero values, accelerating the learning process and simplifying the model.
Regarding the Softmax function, we are going to use it just in the output layer and this depends on the fact that we are approaching a multi-classification problem. Specifically, it is used to standardize outputs converting them into

probabilities that sum to one. It is calculated as softmax(z) = $\frac{\exp(z)}{\sum_j \exp(z_j)}$ and can be seen as a generalization of the *Sigmoid* function [4].

2.1.2 Loss function

The loss function - in this case related to classification problems - is a particular function representing the price paid through the erroneous prediction of the model's labels. They can be of different types but in the context of our essay we are going to use the *Categorical Crossentropy* -specifically designed for multiclassification tasks - to calculate the training and validation loss. It takes the form CCE(p,t) = $-\sum_{c=1}^{C} t_{0,c} \log(p_{0,c})$ where p is an element of the prediction vector, while t is an element of the target one [5]. However, for the sake of the project we are going to measure the test error of the models using a zero-one loss function: it outputs a zero if the prediction is correct and a one otherwise, then computes the mean of the result.

2.1.3 Optimization Algorithms

Optimizers are a very important part of a Neural Network's architecture, they are the algorithms responsible for the minimization of the loss function by changing the learning rate and the weights of the network. In our project we are going to test a wide variety of optimizers, with a major focus (based on the results of the analysis) on the *Stochastic Gradient Descent* (SGD) and a variation of the *Adam* (Adaptive moment estimation), named *Adamax*.
The SGD works calculating the gradient of the loss function moving in the direction which permits to decrease it with respect to a parameter vector θ. Particularly, the stochastic version uses just a random instance of the training set at every step, computing the gradient based only on that single instance. The update method is the following:

$$\theta_{t+1} = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Where η is known as *learning rate* and it has to be chosen so that it is not to small, because this could cause a too slow convergence process, but also not too high, since that could cause divergence. The characteristics of the

SGD permit to decrease the overall computation complexity, speeding up the analysis with respect to the normal Gradient Descent.

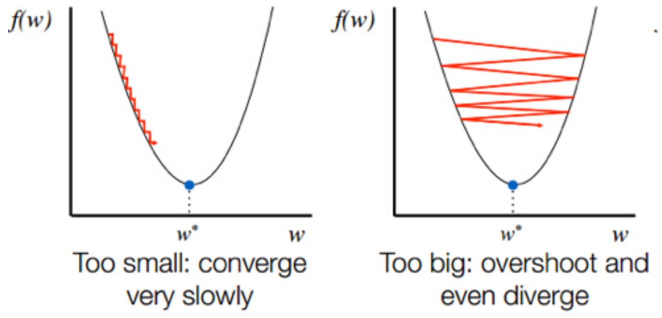

f(w)
w*
w
Too small: converge very slowly
f(w)
w*
w
Too big: overshoot and even diverge

Figure 2: Stochastic Gradient Descent - Learning Rate

Regarding the Adamax algorithm, it is actually a variant of the Adam [7], and while the least scales the gradient proportionally to the ℓ_2 norm of their individual current and past gradients, the first one uses the ℓ_∞ norm obtaining the following update function:

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

where $\hat{m}_t$ is the bias-corrected first moment estimate, and u_t is the the infinity norm-constrained uncentered variance v_t, and is equal to

$$u_t = \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)|g_t|^\infty = \max(\beta_2 \cdot v_{t-1}, |g_t|)$$

.

2.2 Convolutional Neural Networks

The Convolutional Neural Networks [8] are a specialized variety of neural networks, generally composed by three types of layers: convolution, pooling, and fully connected layers. The first two layers which are not present in the standard networks, are used to extract the (hopefully) most important features from the images, decreasing the amount of computational power needed and improving the overall efficiency of the model.

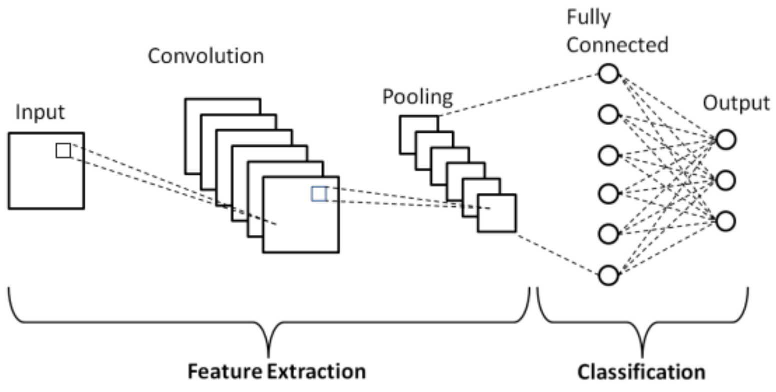


Figure 3: An Example of Convolutional Neural Network

The *convolution layer* is used to apply the homonym *convolution*, which is a linear operation that in our case is carried out between the input matrix and a smaller matrix of weights, named *filter* (or kernel). The filter is applied to each overlapping part of the input data, left to right, top to bottom and generates a resulting "feature map". Its use across an entire image permits to detect a specific feature anywhere in it.

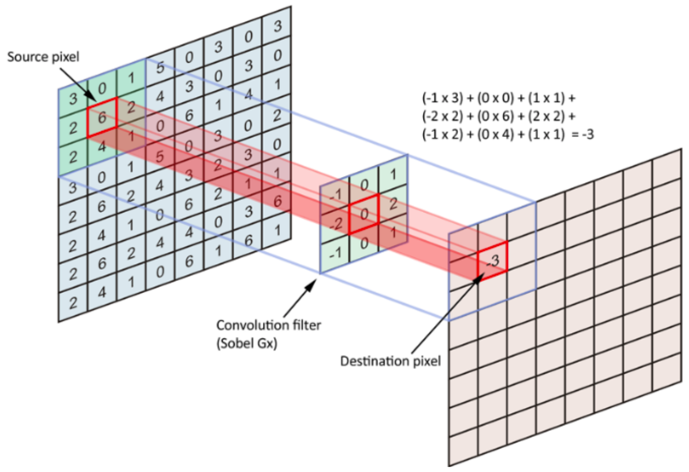

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 4: Example of Convolution Layer

Furthermore, the *pooling layer* is used to reduce the data dimensionality by combining the outputs of neuron clusters of one layer into a single neuron in the next layer. Generally a small cluster of the input, like 2x2, is chosen. The choice of the neurons to maintain in each cluster can be made following different strategies like the *Max Pooling* which selects the highest value, or the *Average Pooling* which keeps an average of the values.

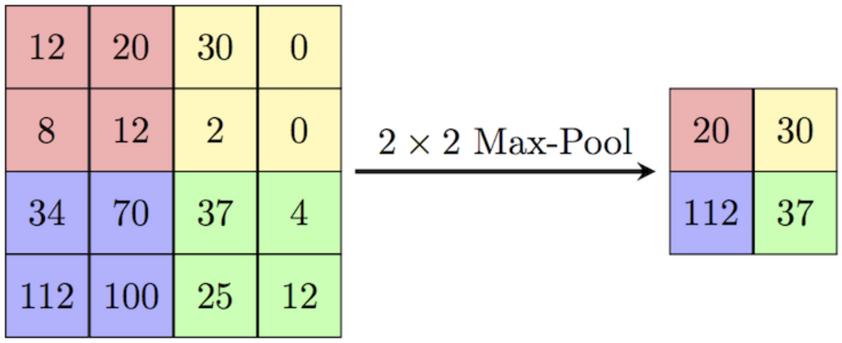


Figure 5: Max Pooling Example

The combined use of these tools permits to obtain really interesting results under the overall performance aspect, primarily on data characterized by a grid-like topology, like in our case with images.

3 Analysis

3.1 Preprocessing of the data set

The data set is formed by a number of images which need to be preprocessed before using them to feed our models. The images are first of all downloaded and separated into to 10 most common varieties, then are converted from a size of 100x100x3 to a smaller one of 32x32x3.

[add number of fruits per category]

Next, the images, already split in training and test set, has been normalized dividing by 255, because the pixel intensity of each color channel is represented as a byte which goes from 0 to 255. In this way we obtain floats ranging from 0 to 1.

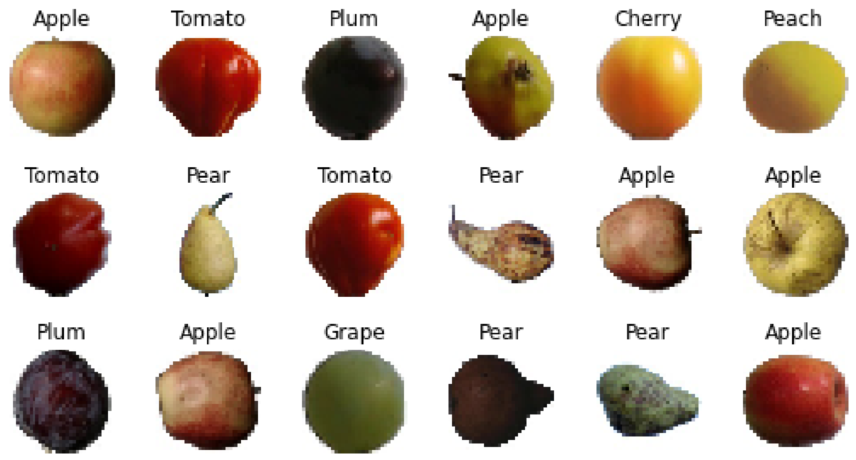


Figure 6: An example of the fruit images

Finally, a function to convert a vector of integers into a binary class matrix is applied to the labels of the different sets.

3.2 Deep Neural Network

The goal of the analysis is to provide many kinds of architectures and to measure the differences between them; the strategy used to provide that is to start with some basic structures moving forward to apply an *hyperparameter* *tuning* process to automate the task and compare the different results. This kind of procedure is necessary because - as said above, in the Introduction of the essay - the choice of the optimal structure is not trivial for a neural network and also an open research sector.

The first structure attempted is the following:

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 1000)	3073000
dense (Dense)	(None, 400)	400400
dense_1 (Dense)	(None, 10)	2310
Trainable params: 3,477,410		
Non-trainable params: 0		

To this “simple” shallow model, many different addition are going to be made, in order to improve the performance step-by-step.

The first layer is used to flatten our input which is converted from 32x32x3 to 3072, the other dense (fully connected) hidden layers contain 1000 and 400 nodes respectively, while the last one is the output layer which contains one node for each classification category, in this case 10. The stochastic gradient descent optimization algorithm has been used with its standard setting. As regularization method, the *early stopping* has been applied, in order to reduce the probability of overfitting applying a patience value of 5 and using the validation accuracy as control benchmark. The 0-1 Loss on the test set returned in this case a value of **0.05584**.

Next, the model has been modified using an exponentially decaying learning rate of the kind

$$\eta_{t+1} = \eta_t * e^{\text{-(decay rate * n. epochs)}}$$

where the number of epochs is an hyperparameter that defines the number times that the learning algorithm will work through the entire training data set and in our case is equal to 30. The decay rate is set at 1e−6 and the starting learning rate is 0.01. Also the *Momentum* (with a generally accepted value of 0.9) and the *Nesterov Accelerated Gradient* [9] are applied to improve the performance [10].

The performance of the 0-1 Loss is now **0.03961**, so the new setting slightly improved the optimization process.

Following our attempt to improve, we are going to try two different models adding some regularizers to our network. In the first case we add two *Dropout Layers* [11] after each hidden layer. They are used to add a probability to exclude each neuron with an arbitrary probability that we fixed at 0.1 and 0.2. In the second case we add an ℓ_1 and ℓ_2 regularization with two penalty terms of 0.1. The final result is not so encouraging and the 0-1 loss increased to **0.04832** and to **0.80433** (!) respectively.

3.3 Hyperparameter tuning

In this section we are going to approach our problem tuning the hyper-parameters trying to find the best model (among those we have searched for). First of all we are going to use our previous setting testing different optimizers: Adam, Rmsprop, Adamax and Nadam (excluding SGD which will be included in the next tuning approach), choosing the one with the best result:

Table 2: Accuracy of the four optimizers

Optimizer	Average Accuracy
Adam	0.76834
Rmsprop	0.35529
Adamax	**0.99977**
Nadam	0.19681

The best optimizer found is the Adamax, briefly described in the previous section of the essay, so it is going to be used in the next tuning process together with the Stochastic Gradient Descent.

Now the structure of each network is singularly optimized with respect to the number of layers, nodes and regularizers. Then, the best models are then saved and used to calculate the 0-1 Loss of the test set. The results are the following:

1. For the SGD optimizer the best parameters are ‘dropout’: 0, ‘l1’: 0, ‘l2’: 0, ‘nl1’: 2, ‘nl2’: 1, ‘nl3’: 3, ‘nn1’: 1500, ‘nn2’: 1500, ‘nn3’: 250 and the final 0-1 Loss is **0.03576**, which is an improvement, but not a much better than the result with respect with our shallow model. Also in this case the regularization parameters has been confirmed to zero.

Layer (type)	Output Shape	Param #
flatten_71 (Flatten)	(None, 3072)	0
dense_363 (Dense)	(None, 1500)	4609500
dense_363 (Dense)	(None, 1500)	2251500
dense_363 (Dense)	(None, 1500)	2251500
dense_363 (Dense)	(None, 250)	375250
dense_363 (Dense)	(None, 250)	62750
dense_363 (Dense)	(None, 250)	62750
dense_363 (Dense)	(None, 10)	2510
Trainable params: 9,615,760		
Non-trainable params: 0		

2. For the Adamax optimizer the parameters are ‘dropout’: 0.1, ‘l1’: 0, ‘l2’: 0, ‘nl1’: 2, ‘nl2’: 0, ‘nl3’:1, ‘nn1’: 1000, ‘nn2’: 1000, ‘nn3’: 1000 and the final 0-1 Loss is **0.07345**.

Layer (type)	Output Shape	Param #
flatten_122 (Flatten)	(None, 3072)	0
dense_695 (Dense)	(None, 1000)	3073000
dropout_375 (Dropout)	(None, 1000)	0
dense_696 (Dense)	(None, 1000)	1001000
dropout_376 (Dropout)	(None, 1000)	0
dense_697 (Dense)	(None, 1000)	1001000
dropout_377 (Dropout)	(None, 1000)	0
dense_698 (Dense)	(None, 10)	10010
Trainable params: 5,085,010		

Layer (type)	Output Shape	Param #
Non-trainable params: 0		

3.4 Convolutional Neural Networks

The analysis related to the CNNs is going to be a little different, because we are going to use some *famous* architectures to measure their performance on our model.

3.5 VGG-16

The VGG-16 [12][13] is a CNN used in 2014 into the ILSVRC competition [14]. The structure is the following:

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 32, 32, 64)	1792
conv2d_27 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_10 (MaxPooling)	(None, 16, 16, 64)	0
conv2d_28 (Conv2D)	(None, 16, 16, 128)	73856
conv2d_29 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_11 (MaxPooling)	(None, 8, 8, 128)	0
conv2d_30 (Conv2D)	(None, 8, 8, 256)	295168
conv2d_30 (Conv2D)	(None, 8, 8, 256)	590080
conv2d_30 (Conv2D)	(None, 8, 8, 256)	590080
max_pooling2d_12 (MaxPooling)	(None, 4, 4, 256)	0
conv2d_33 (Conv2D)	(None, 4, 4, 512)	1180160
conv2d_33 (Conv2D)	(None, 4, 4, 512)	2359808
conv2d_33 (Conv2D)	(None, 4, 4, 512)	2359808
max_pooling2d_13 (MaxPooling)	(None, 2, 2, 512)	0
conv2d_36 (Conv2D)	(None, 2, 2, 512)	2359808

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 2, 2, 512)	2359808
conv2d_36 (Conv2D)	(None, 2, 2, 512)	2359808
max_pooling2d_14 (MaxPooling)	(None, 1, 1, 512)	0
flatten_2 (Flatten)	(None, 512)	0
dense_6 (Dense)	(None, 4096)	2101248
dense_7 (Dense)	(None, 4096)	16781312
dense_8 (Dense)	(None, 10)	40970
Trainable params: 33,638,218		
Non-trainable params: 0		

This convolutional network, while using a classical structure mixing convolution and pooling, is deep and able to better detect and identify patterns in the images, this leads a big improvement in performance, since the 0-1 Loss decreases to **0.01311**.

3.6 Res-Net 34

The Res-Net 34 [15][16] is the winner of the ILSVRC 2015 and has the following structure:

Layer (type)	Output Shape	Param #
conv2d_112 (Conv2D)	(None, 16, 16, 64)	9408
batch_normalization_73	(None, 16, 16, 64)	256
activation_3 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_18 (MaxPooling)	(None, 8, 8, 64)	0
residual_unit_32 (ResidualUnit)	(None, 8, 8, 64)	74240
residual_unit_33 (ResidualUnit)	(None, 8, 8, 64)	74240
residual_unit_34 (ResidualUnit)	(None, 8, 8, 64)	74240
residual_unit_35 (ResidualUnit)	(None, 4, 4, 128)	230912
residual_unit_36 (ResidualUnit)	(None, 4, 4, 128)	295936
residual_unit_37 (ResidualUnit)	(None, 4, 4, 128)	295936
residual_unit_38 (ResidualUnit)	(None, 4, 4, 128)	295936

Layer (type)	Output Shape	Param #
residual_unit_39 (ResidualUnit)	(None, 2, 2, 256)	920576
residual_unit_40 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_41 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_42 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_43 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_44 (ResidualUnit)	(None, 2, 2, 256)	1181696
residual_unit_45 (ResidualUnit)	(None, 1, 1, 512)	3676160
residual_unit_46 (ResidualUnit)	(None, 1, 1, 512)	4722688
residual_unit_47 (ResidualUnit)	(None, 1, 1, 512)	4722688
global_average_pooling2d_2	(None, 512)	0
flatten_5 (Flatten)	(None, 512)	0
dense_11 (Dense)	(None, 10)	5130
Trainable params: 21,289,802		
Non-trainable params: 17,024		

The Res-Net - as the name suggest - is based on the presence of the *Residual Layer* which is characterized by a “shortcut connection” which permits to the information to go deeper in the network skipping one ore more layers. This is useful in order to avoid the *vanishing gradient* problem which is due to the fact that many activation functions, like the sigmoid for example, have a derivative that approaches zero when the input becomes large (or small). This issue can cause a “degradation” in the overall training accuracy of a model with an high number of layers.

There are many possible solutions to that, like using normalized initialization or intermediate normalization layers (as suggested by the authors of the Res-Net paper [15]), but the construction of a network with residual layers has been proven highly effective.

The 0-1 Loss of **0.00926** - the highest obtained - confirmed the efficiency of this kind of architecture for our image recognition task.

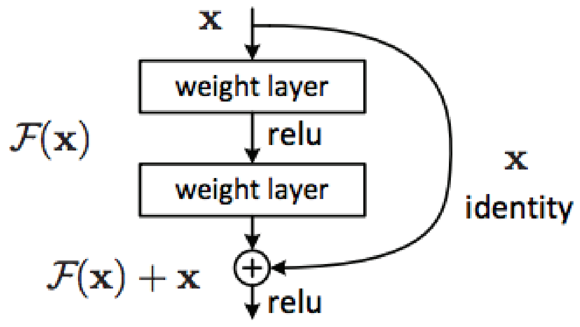

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 7: Example of a Residual Layer

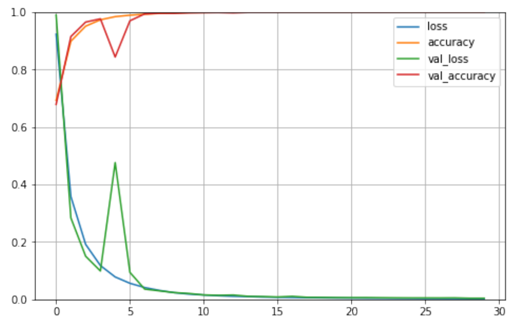
4 Conclusions

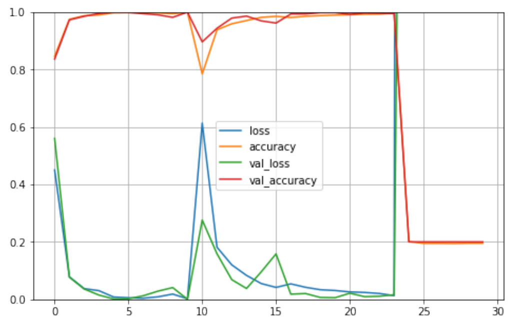
The step by step optimization process of the Deep Feedforward Neural Networks permitted to achieve quite good classification performances, with a minimum loss - obtained by the tuned Network with the Stochastic Gradient Descent Optimizer - of 0.03576. The result shows an improvement of the 9.72% with respect to the shallow network.

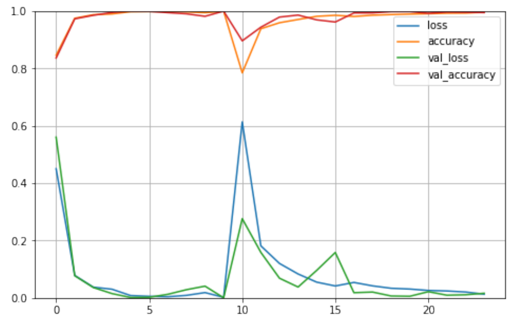
However a structured Convolutional Neural Network - as we saw in the last section of the essay -can easily outperform these results, thanks to the fact that those models are optimized for Image Recognition tasks, thus able to detect many more patterns in the input data. The best performance has been reached by the Res-Net 34 with a 0-1 Loss of 0.00926 with a reduction of the 74.11% with respect to the tuned network using Stochastic Gradient Descent and of the 29.37% relatively to the VGG-16.

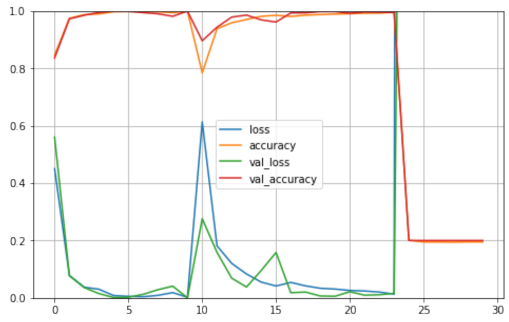
5 Plots

From left to right there are the graphs of the first four Feedforward Deep Neural Network models:









The following are the graphs of the two tuned architectures:

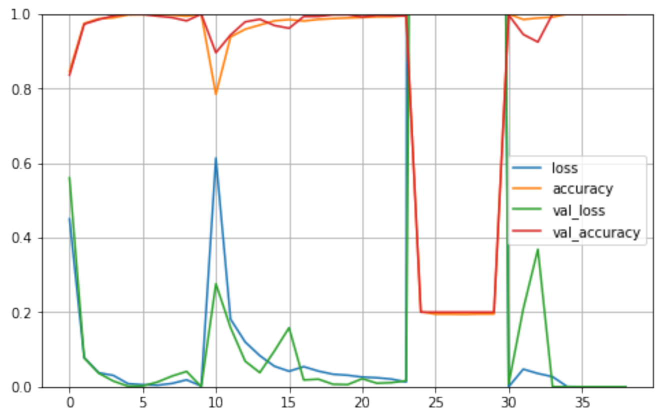


Figure 8: Stochastic Gradient Descent

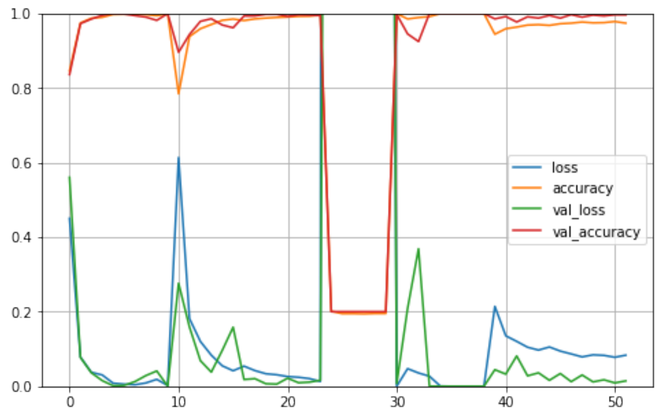


Figure 9: Adamax

Finally, the graphs of the two Convolutional Neural Networks Networks:

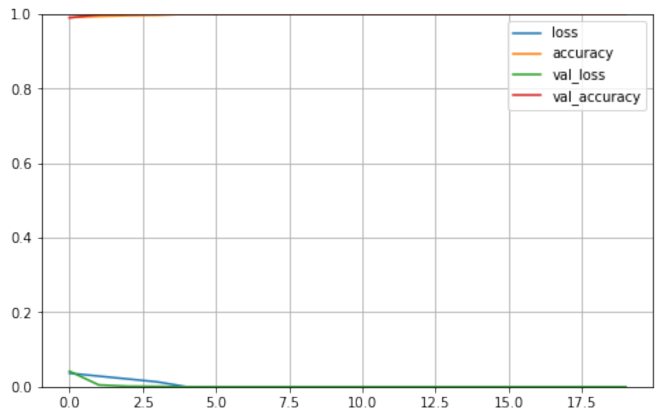


Figure 10: VGG-16

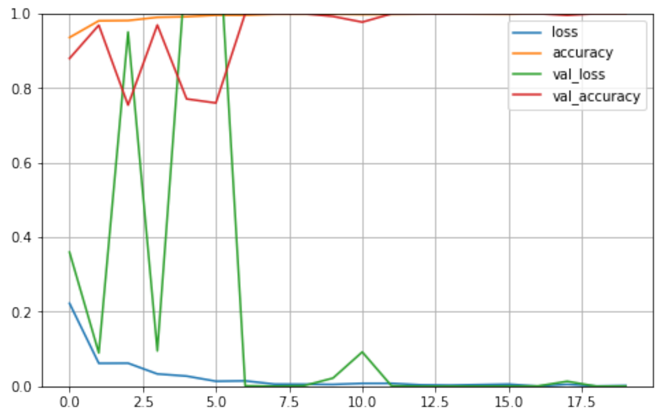


Figure 11: Res-Net 34

6 Bibliography

[1] Kaggle Data set

[2] “A Logical Calculus of the Ideas Immanent in Neuron Activity” - W. S. McColluch and W. Pitts (1943)

[3] “Deep Learning” - I. Goodfellow, Y. Bengio and A. Courville (2016), pag. 187

[4] “Deep Learning” - I. Goodfellow, Y. Bengio and A. Courville (2016), pag. 178-179

[5] How to use binary & categorical crossentropy with TensorFlow 2 and Keras?

[6] “Gradient Descent based Optimization Algorithms for Deep Learning Models Training” - J. Zhang (2019)

[7] “Adam: A Method for Statistic Optimization” - D. P. Kingma, J. L. Ba (2014)

[8] “Generalization and Network Design Strategies” - LeCun (1989)

[9] “A Method for Unconstrained Convex Minimization Problem with the Rate of Convergence $O(\frac{1}{k^2})$” - Y. Nesterov (1983)

[10] “Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow” - A. Géron (2019), pag. 351-353

[11] “Dropout: A Simple Way to Prevent Neural Networks from Overfitting” - N. Srivastava Et Al. (2014)

[12] “Very Deep Convolutional Networks for Large-Scale Image Recognition” - K. Simonyan and A Zisserman (2014)

[13] Step by step VGG16 implementation in Keras for beginners

[14] Image-Net

[15] “Deep Residual Learning for Image Recognition” - K. He Et Al. (2015)

[16] “Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow” - A. Géron (2019), pag. 478-479