Intelligent Systems

Gatto Marco - Mat. [730847]

Laino Lorenzo - Mat. [731124]

28 January 2021

# **Introduction**

The scope of this project concerns the study and the practical application of some machine learning, and in particular deep learning, techniques.

The project mainly focuses on a dataset called “rock\_paper\_scissors”, available on: <https://www.tensorflow.org/datasets/catalog/rock_paper_scissors>. The dataset contains 2892 images of hands playing the rock, paper and scissor game.

The experiment is developed using the tools and resources offered by the Python Environment, and in particular: Tensorflow, Keras. OpenCV and more.

The experiment is designed to investigate the behaviours and the capabilities of different Neural Networks under different experimental conditions. In particular, the first part of the experiment uses some pre-trained Convolutional Neural Networks (CNNs) in order to classify the dataset images. The second part, instead, shows the steps of the creation from scratch of a CNN. The obtained results are compared at the end of each part.

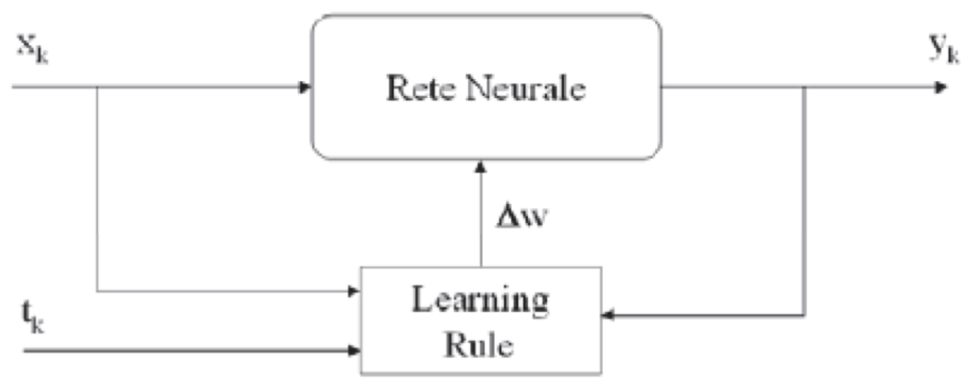
The report is divided into two parts:

1. The first one introduces the main theoretical concepts that has been applied.
2. The second one describes the steps applied during the experimental part and the obtained results.

# **Theoretical Part**

## Supervised Learning

Supervised learning tries to model the relationship between measured features of data and some label associated with the data. Once this model is determined, it can be used to apply labels to new, unknown data. This is further subdivided into classification tasks and regression tasks: in classification, the labels are discrete categories, while in regression, the labels are continuous quantities. The experiment in this project concerns a classification problem.



(Xk, tk) training sample with input-output pairs

## Deep Learning

Deep learning means using a neural network with several layers of nodes between input and output. The series of features between input & output do feature identification.

A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into a more abstract representation (e.g., edge → nose → face). The output layer combines those features to make predictions.

Backpropagation is the widely used learning algorithm for training ordinary neural networks. It aims to identify an optimal set of weights able to minimize the error when processing training data.

In order to solve the problem related to the hidden layers’ node weights, the Backpropagation algorithm analyzes the error in the final layer and then looks back at how it is distributed in the previous layers. So, the information flow goes forward and then the computed error statement is back-propagated updating weights.

In particular, the main steps followed by the algorithms are:

* The process starts with an arbitrary, but not all equal, set of weights throughout the network.
* The application of the generalized delta rule at any iterative step involves three basic phases:
  1. A training vector is presented to the network and is allowed to propagate through the layers to compute the outputs for each node.
  2. The outputs of the nodes in the output layer are then computed against their desired responses to generate the error terms.
  3. The appropriate error signal is passed to each node and the corresponding weight changes are made according to a backward pass through the network.

In a successful training session, the network error decreases with the number of iteration and the procedure converges to a stable set of weights that exhibits only small fluctuations with additional training.

Epoch is an important concept while dealing with the training of a neural network. It is the iterative processing of the training set. Usually, the learning process involves several epochs and the termination condition is formulated in terms of the minimum error accepted or in terms of the number of epochs.

## Convolutional Neural Network (CNN)

Convolutional Neural Networks are very similar to ordinary Neural Networks:

* They are made up of neurons that have learnable weights and biases.
* Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.

The difference between a CNN and an ordinary Neural Network (such as MLP) is about the connections between the layers. In CNN not all layers are fully connected and in ordinary Neural Network all layers are fully connected. This difference makes the CNN the best alternative to work with images.

CNN have a topology that organizes 3D volumes of neurons to be oriented to process images. Neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully connected manner.

## Tensorflow

TensorFlow is an open-source software library released in 2015 by Google to make it easier for developers to design, build and train deep learning models. TensorFlow was originated as an internal library that Google developers used to build models in-house, and we expect additional functionality to be added to the open-source version as it is tested and vetted in the internal flavor. Although TensorFlow is only one of several options available to developers, we choose to use it here because of its thoughtful design and ease of use.

On a high level, TensorFlow is a Python library that allows users to express arbitrary computation as a graph of data flows. Nodes in this graph represent mathematical operations, whereas edges represent data that is communicated from one node to another. Data in TensorFlow is represented as tensors, which are multidimensional arrays (representing vectors with a 1D tensor, matrices with a 2D tensor, etc.).

Although this framework for thinking about computation is valuable in many different fields, TensorFlow is primarily used for deep learning in practice and research.

## Keras

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Keras has the following key features:

* Allows the same code to run on CPU or on GPU, seamlessly.
* User-friendly API which makes it easy to quickly prototype deep learning models.
* Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
* Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

## OpenCV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. All the OpenCV array structures are converted to and from Numpy arrays. This makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

## Matplotlib

Matplotlib is a multiplatform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of Matplotlib’s most important features is its ability to play well with many operating systems and graphics backends. Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish.

## Numpy

NumPy (short for Numerical Python) provides an efficient interface to store and operate on dense data buffers. In some ways, NumPy arrays are like Python’s built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size.

## SciKit-Learn

Scikit-learn is an open-source machine learning library that supports supervised and unsupervised learning in Python. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.

# **Experimental Part**

”Rock\_paper\_scissors” dataset contains 2892 colour images, each with dimensions *300x300px*. The content of the images within the dataset is sampled from 3 classes:

* Rock
* Paper
* Scissors



### Process the data

The deep learning Keras library provides direct access to the ”Rock\_paper\_scissors” (Keras dataset) with relative ease, through its dataset module.

Training Dataset (train\_ds): this is the group of our dataset used to train the neural network directly. Training data refers to the dataset partition exposed to the neural network during the training.

Test Dataset (test\_ds): this partition of the dataset evaluates the performance of our network after the completion of the training phase.

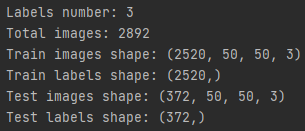
Since the experiment is supervised, both Training and Test Dataset are composed by images and labels. For this reason, it is necessary to define a method to split the images from the labels.Graphical user interface, text

Description automatically generated

Given a dataset and a dimension value as input, the method generates two arrays containing the images and the labels respectively. Furthermore, the method uses the dimension value and the cv2 library to resize the images as *dimension x dimension*.

In order to obtain a faster and easier model, the images have been resized to *50x50px*.

At the end of the data processing phase, an overview of the results obtained from the previous steps is shown.



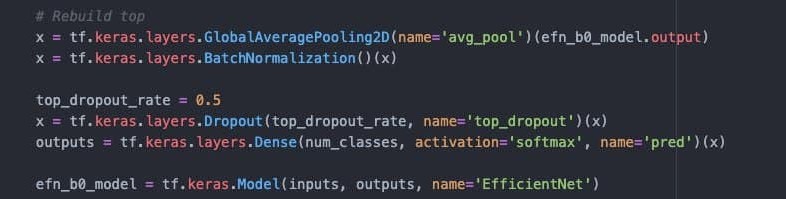
## EfficientNetB0

EfficientNet-B0 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images.

### Build the model

Since EfficientNet-B0 is a well-known pre-trained model, we can import it from the Keras library. Furthermore, it is needed to create an additional layer that is used as the new input layer of the model. The input size, corresponding to the dimension value of each image, is provided to this layer.

The weights of the pre-trained model need to be frozen. In this way, it is possible to reuse all the discriminant capability learnt by the model during the previous training phases.

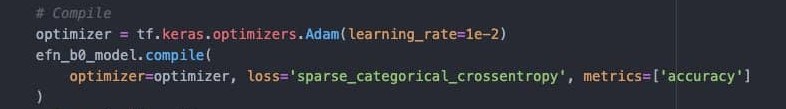
Finally, it is needed to re-build the top layers of the model. These are the only layers that will be trained in the following phases.

In particular, the introduced layers are:

* GlobalAveragePooling2D: it is the Keras representation of an average pooling layer. Average pooling is a variant of sub-sampling where the mean of the pixels that fall within the receptive field of a unit within a sub-sampling layer is taken as the output. The pool size is still set to the size of the input layer.
* BatchNormalization: it is a technique that mitigates the effect of unstable gradients within a neural network through the introduction of an additional layer that performs operations on the inputs from the previous layer. The operations standardize and normalize the input values, after that the input values are transformed through scaling and shifting operations.
* Dropout: it is a technique that works by randomly reducing the number of interconnecting neurons within a neural network. At every training step, each neuron has a chance of being left out, or rather, dropped out of the collated contributions from connected neurons. In the above piece of code, the probability of being left out equals 0.5.
* Dense: it is the Keras representation of a Fully-Connected layer. It has an embedded number of arbitrary units/neurons within. Each neuron is a perceptron. This layer represents the output layer: indeed, its number of units equals the number of different classes of the dataset. Softmax is a type of activation function that is utilized to derive the probability distribution of a set of numbers within an input vector. The output of a softmax activation function is a vector in which its set of values represents the probability of an occurrence of a class or event. The values within the vector all add up to 1.

At the end, the final model is built-up.

### Compile the model

Before the training phase, the model is compiled. In particular, it is needed to specify some parameters, such as the optimizer, the loss function and the metrics.

The optimizer is the algorithm used to change the attributes of the neural network such as weights and learning rate to reduce the losses, and it is used to solve optimization problems by minimizing the function.

The learning rate is a factor value that determines the level of updates that are made to the values of the weights of the network. It is a type of hyperparameter.

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. In particular, this method is computationally efficient, invariant to diagonal rescaling of gradients, has little memory requirement and is well suited for problems that are large in terms of data and parameters.

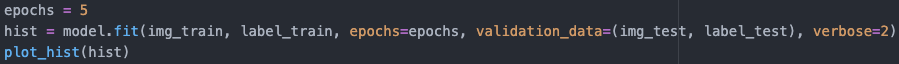
The loss function is a method that quantifies ‘how well’ a machine learning model performs; the quantification is a cost based on a set of inputs, which are referred to as parameter values. The parameter values are used to estimate a prediction, and the ‘loss’ is the difference between the predictions and the actual values.

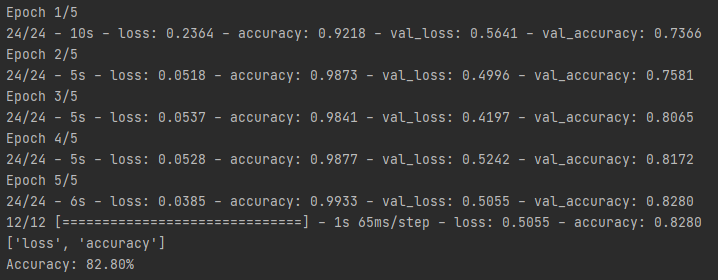
The metrics are used to measure the performances of the model. The accuracy is the ratio of number of correct predictions to the total number of input samples.

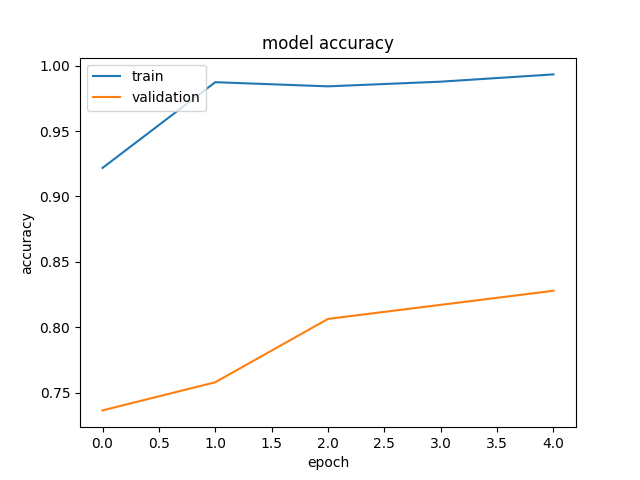
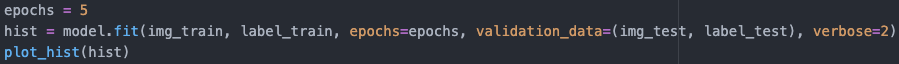


The image above shows the complete method used in this experiment in order to build-up and compile the model.

### Train the model

In order to train the model, the number of epochs and the validation set should be defined.

The model is trained along 5 epochs and the test set is also used as validation test. An example of training execution is shown in the following image.

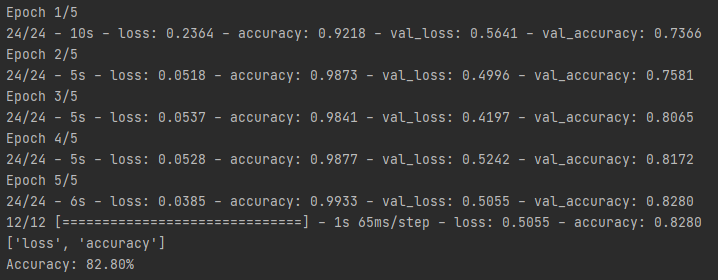
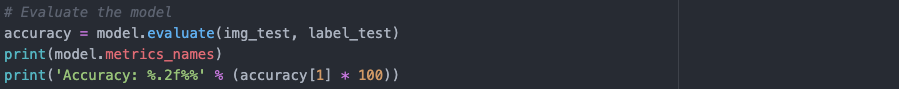
It is also possible to show the training and validation accuracies with a graphical representation.

The plot graphically represents the model fitting execution of the previous example. In particular, it can be seen that both the training and the validation accuracy grow in a constant way. However, the validation accuracy never reaches very high values of accuracy.

A more detailed analysis of the results will be performed in the conclusion paragraph through the comparison of the results got from different executions.

(eventually, use validation set (taken from train) instead of test set in the fit validation)

### Evaluate the model

This phase is needed to evaluate the performances of the model. In particular, the metrics used to measure the performances are the ones specified when the model is compiled. In our case, the reference metric is the accuracy, that is defined as the ratio of the number of correct predictions to the total number of input samples.

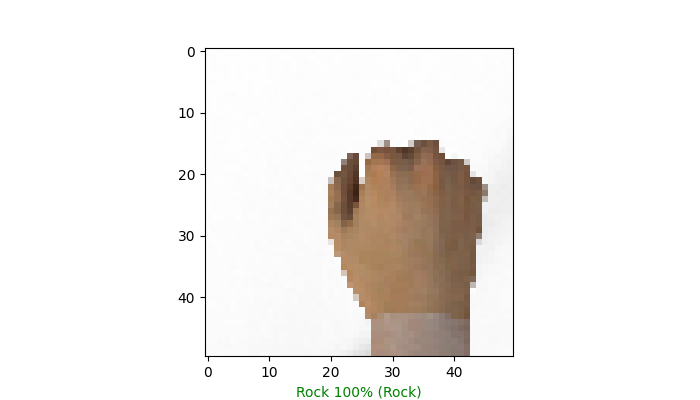
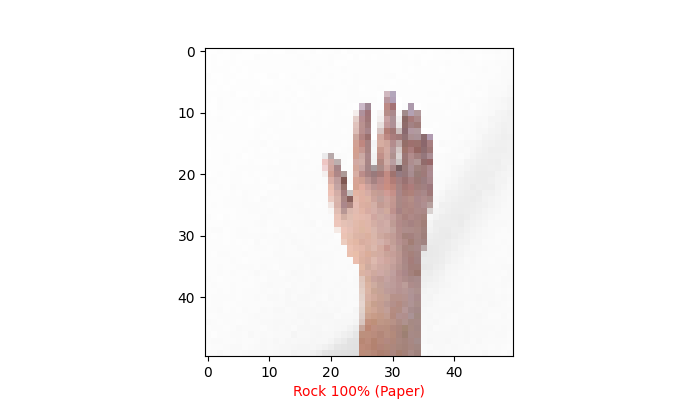
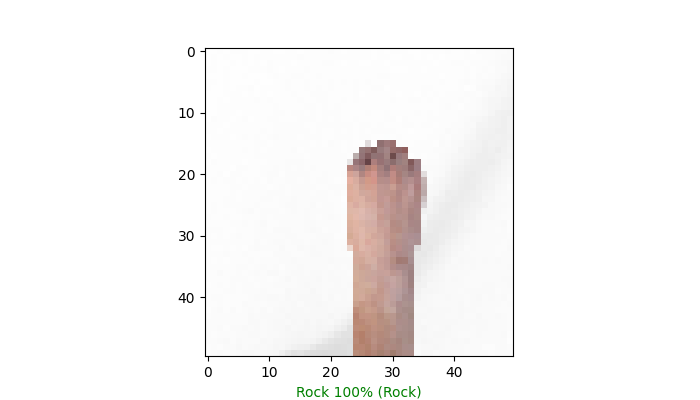
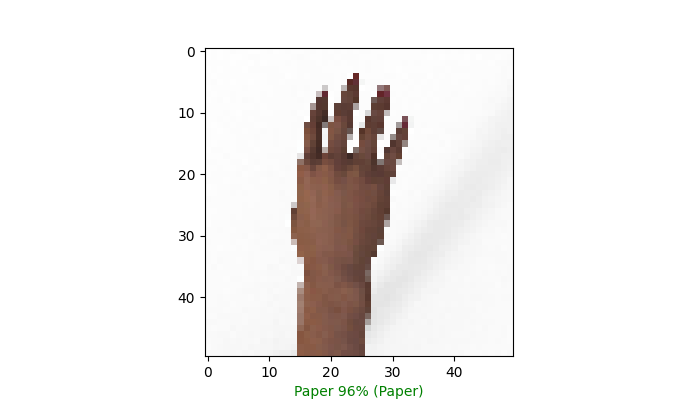
Since the validation set used during the training phase is the same as the test set, the value of the final accuracy equals the final value in the previous plot.

### Make predictions

The final phase concerns the usage of the model to classify some images that have not been used during the training phase. In particular, the model is asked to predict the label of all the images contained in the test set.

Furthermore, it is needed to create a method that shows the results of the prediction on some randomically-chosen images.

This method simply choses 4 random indexes, takes from the test set the images corresponding to the chosen index and attaches to the images both the corresponding label and predicted values.



Since the output layer of the model uses the 'softmax' activation function, a probability value is associated with each input image and it represents the probability that the specific image is classified from the model with that specific label.

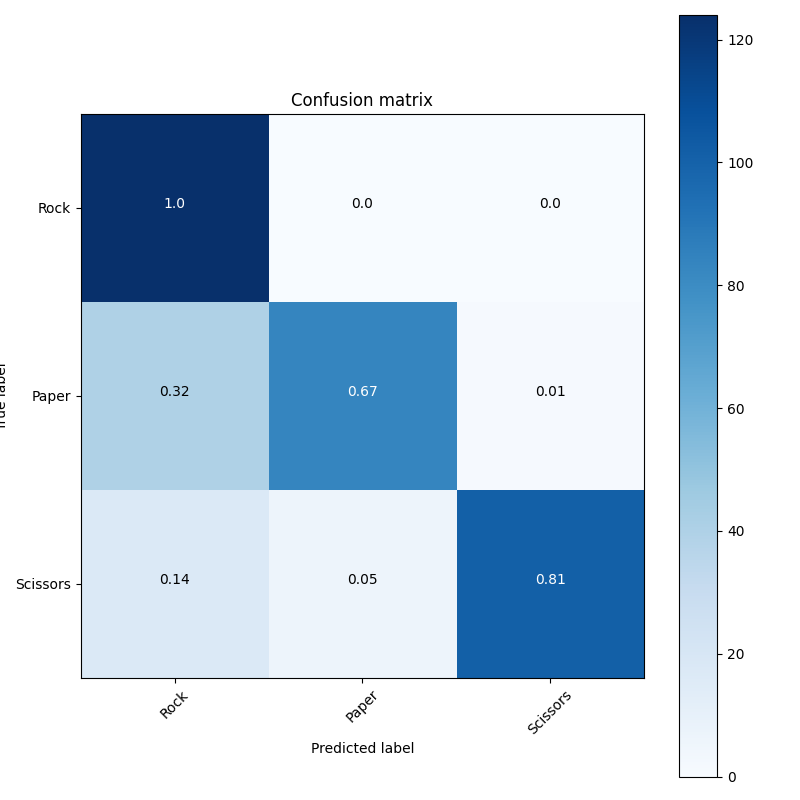
We also compute and show the confusion matrix concerning the labels predicted from the model. For this reason, it is used the utility provided from the library SciKit-Learn that automatically computes the confusion matrix given both the true and the predicted labels.

<code 1>

Furthermore, it is needed to define a method that, given the confusion matrix, normalizes its values and then shows it as a plot.

<code method>

An example of the results obtained through this code is the following:



Even in this case, the image represents the confusion matrix of the model used as example in the previous steps.

### Conclusion

…