Universitá degli studi di Milano

M.Sc. Data Science & Economics

Cloud Computing and Algorithms for Massive Data

Alessia Leo Folliero
Code: 08399A

October 23, 2023

Contents

1 Introduction

This project is part of the Cloud Computing and algorithms for Massive Data exam. The aim of this project is to use Convolutional Neural Network in order to classify leaves species. In this paper you will find the way in which the task has been implemented the CCN and the results I obtained.

2 CCN theory

Neural Networks are algorithms modeled after our brains that try to simulate human biological neural network.
Convolutional Neural Network are a special kind of Neural Network that has at least one convolution layer. We usually use CNN because they are helpful if we want to obtain local information (feature), to reduce the overall complexity of the model and because it needs just few prepossessing of the images.

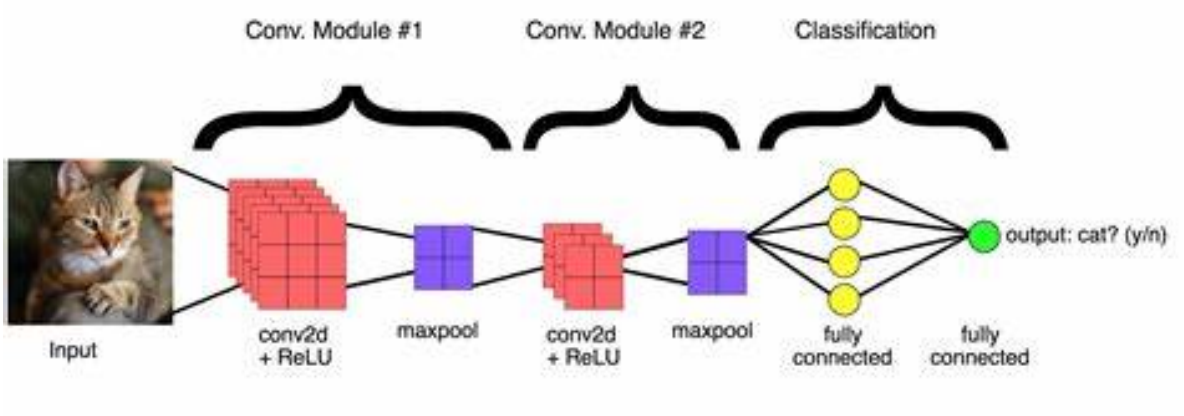

Figure 1: Example of CCN.

A CCN is composed by three main layers: Convolutional, pooling and fully connected layer.

- **Convolutional Layer**: The filter is applied to our input to extract its features. A filter is applied to the image multiple times and create a feature map that helps to classify the image.
- **Pooling layer**: The aim of this layer is to reduce the dimensions of the feature map which help in preserving the important information of the input image and reduce the computation time. It works similarly to the convolution layer but the function that is applied to the kernel and image is not linear. The most common pooling functions are: Max pooling or average pooling.
- **fully connected layer**: This layers connects the information extracted from previous layers and classifies the input with the corresponding label.

In between the second step (pooling) and the third one there is the **Flattening** step.

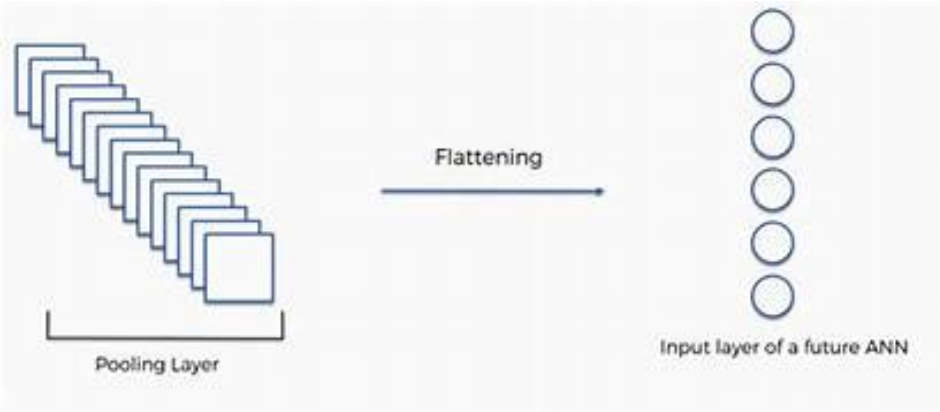

Figure 2: Example of the Flattening step.

In this step we convert data into a one dimensional array.The reason why we do that is because in this way we are able to give it as an input to the Network in the following step.

In order to get the output, after we calculated the weighted sum of all the inputs and weights of the connection we need to add bias to this sum, since image recognition in not a linear operation. In order to introduce this non-linearity we use activation functions. There are serveral type of activation function i.e Sigmoid (Binary classification),ReLu and LReLu. Apart from those layer some other could be added to the CCN such as:

- **Dropout Layer**: In this layer a percentage p of nodes of the neural network is being dropped. This allow to have a new network architecture thanks to which the risk of over fitting is reduced.
- **BatchNormalization**: Is a normalization technique that normalize the output of the previous layer. Thanks to this technique learning becomes more efficient and it can be used as a regularization to avoid overfitting.

3 Dataset

The dataset has been proposed by the professor and can be easily find on kaggle, this is the link to the dataset
http://https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification
This dataset contain a variety of healthy and diseased leaves from different plants such as Mango, Basil, Pomegranate and so on. It contains 4503 images which are composed by 2278 healthy leaves images and 2225 diseased leaves images. The dataset was already splitted into train, validation and test. In this project only the Healthy leaves have been taken into account .

4 Pre-Processing

The pre-processing of the image is very important in order to obtain good results. In this project the pre-processing steps are:

- Delete all the diseased images
- Resize the image as (256,256).
- Convert the image to grey scale.
- ImageDataGenerator from Keras.

Figure 3: Original Healthy Basil leaf.

Figure 4: Transformed image.

5 CNN Model

In this model the hyperparameters have not been fixed but instead for each layer the hyperparameters are chosen thanks to the hyperparameter tuning which is done after the model has been built.
The Convolutional layer has as filter an integer number ranging from 32 to 128 with a step of 16. Moreover for this filter the kernel size has been set to (3,3), the activation to ReLu and the padding parameter to "same".
For the maxpool layer the parameter strides has been set to (2,2) in this way the window size is bigger than the default size. Also in this layer the padding has been set to "same".
The dropout layer has as rate a float that is in rage 0.0 to 0.3 with a 0.05 step.
The dense layer has instead as unit an integer that goes from 32 to 128 with a 16 step.
The compile layer takes three parameters: Optimizer, loss and metric. In this model as the loss functions has been chosen the sparse categorical cross entropy which is used in multi class classification when the labels are encoded as integers. For the optimizer instead the Adam optimizer has been applied, this optimizer adjust the learning rate throughout the training. The metric parameter has been set to "accuracy", in this way when we train the model the we will see the accuracy score on the validation set.

```
#defining model
def My_Cnn(param):
  model=Sequential()

  #adding convolution layer
  model.add(Conv2D(filters=param.Int('conv1_filter', min_value=32, max_value=128, step=16),kernel_size=(3,3),activation='relu',input_shape=(256,256,1),padding="same"))
  #adding pooling layer
  model.add(MaxPool2D(pool_size=(2,2),strides=(2,2), padding='same'))
  #Dropoutlayer
  model.add(Dropout(param.Float('drop_1', min_value=0.10, max_value=0.3, step=0.05)))


  ###Second Layer
  model.add(Conv2D(filters=param.Int('conv2_filter', min_value=32, max_value=128, step=16),kernel_size=(3,3),activation='relu',input_shape=(256,256,1),padding="same"))
  model.add(MaxPool2D(pool_size=(2,2), padding='same'))
  model.add(Dropout(param.Float('drop_2', min_value=0.10, max_value=0.3, step=0.05)))

  ###Third Layer
  model.add(Conv2D(filters=param.Int('conv3_filter', min_value=32, max_value=128, step=16),kernel_size=(3,3),activation='relu',input_shape=(256,256,1),padding="same"))
  model.add(MaxPool2D(pool_size=(2,2), padding='same'))
  model.add(Dropout(param.Float('drop_3', min_value=0.10, max_value=0.3, step=0.05)))

  #adding fully connected layer
  model.add(Flatten())
  model.add(Dense(units=param.Int('units', min_value=32, max_value=128, step=16),activation='relu'))
  model.add(Dropout(param.Float('drop_c', min_value=0.0, max_value=0.3, step=0.05)))

  #adding output layer
  model.add(Dense(11,activation='softmax'))

  #compiling the model
  model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

  return model
```

Figure 5: CNN model implementation.

After the parameter have been tuned this is the result:

```
{'conv1_filter': 80,
 'drop_1': 0.0,
 'conv2_filter': 112,
 'drop_2': 0.2,
 'conv3_filter': 48,
 'drop_3': 0.2,
 'units': 112,
 'drop_c': 0.0}
```

Figure 6: Best hyperparameters.

5.1 Results

In the following plot it can be seen the validation loss and the accuracy loss in one plot and in the other one the validation accuracy and train accuracy. In the first plot those element are shown after 10 epochs and in the second one after 30 epochs. For the 30 epochs procedure an early stopping rule was added in order to prevent overfitting.

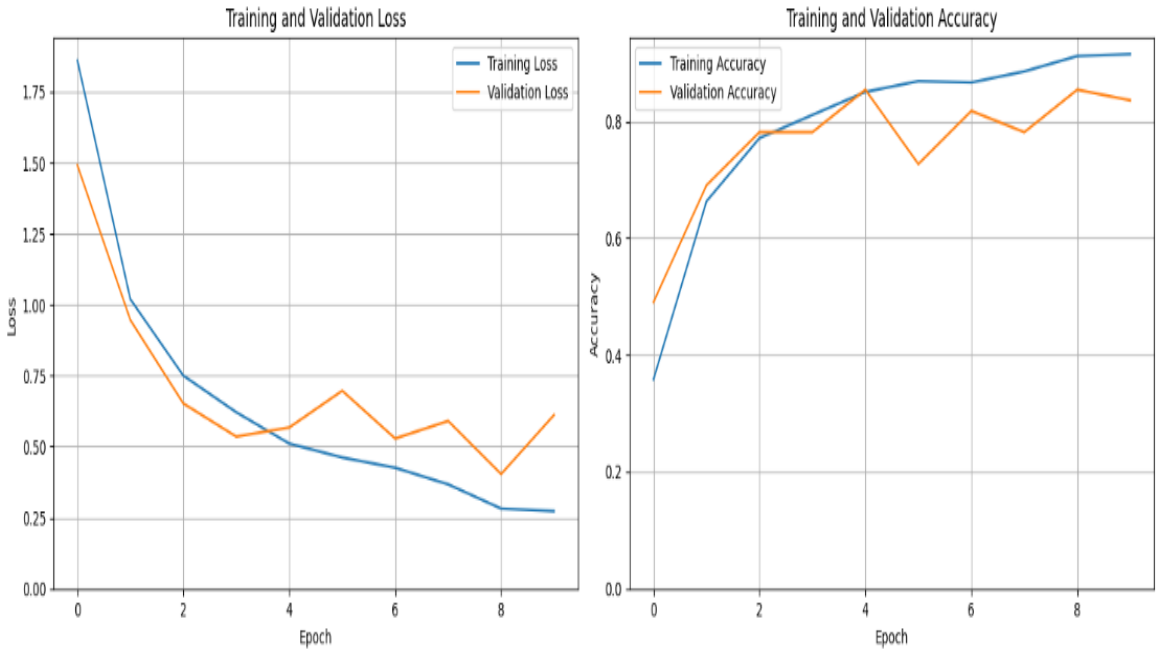

Figure 7: Plot after training with 10 epochs.

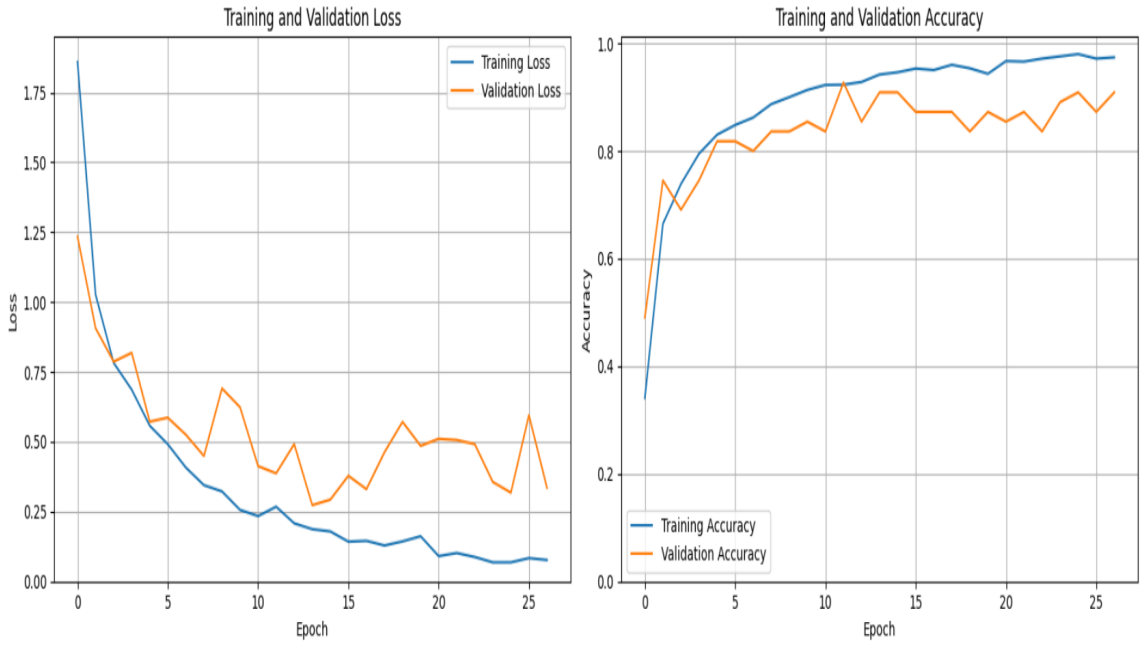

Figure 8: Plot after training with 30 epochs.

From the plot it can be seen that training and validation loss seems to have the same downward trend from this we can understand that the model is learning. From the 12 epochs on the two curves seems to diverge a little and are not stabilized jet, this means that there is further space for improvement so we should give the model more training time.This kind of behavior could be also motivated by the fact that the validation data has only 55 images.
Moving on to the interpretation of the train and validation it can be seen that also in this case the behavior of the two curves is the same, meaning that the algorithm is learning. From the plot we are not able to see a convergence between the two curves but they seem to point in the same direction which is a good sign that the model is performing well.

The prediction results of the model used in figure 8 are displayed in the following table 1

Item	Score
Loss	0.25108
Accuracy	0.94545

Table 1: Result of the evaluation.

From the table it can be seen that the accuracy score that we obtained is close to 95% while instead the loss is 25

5.2 Performance plot

In this plot it can be seen the performance of the model on both training and validation data.
The curves represent the value of the metric at a specific epoch. In order to have a better interpretation a red line is drawn in the plot. This red line represent the point of intersection between the two curves along with the optimal value.

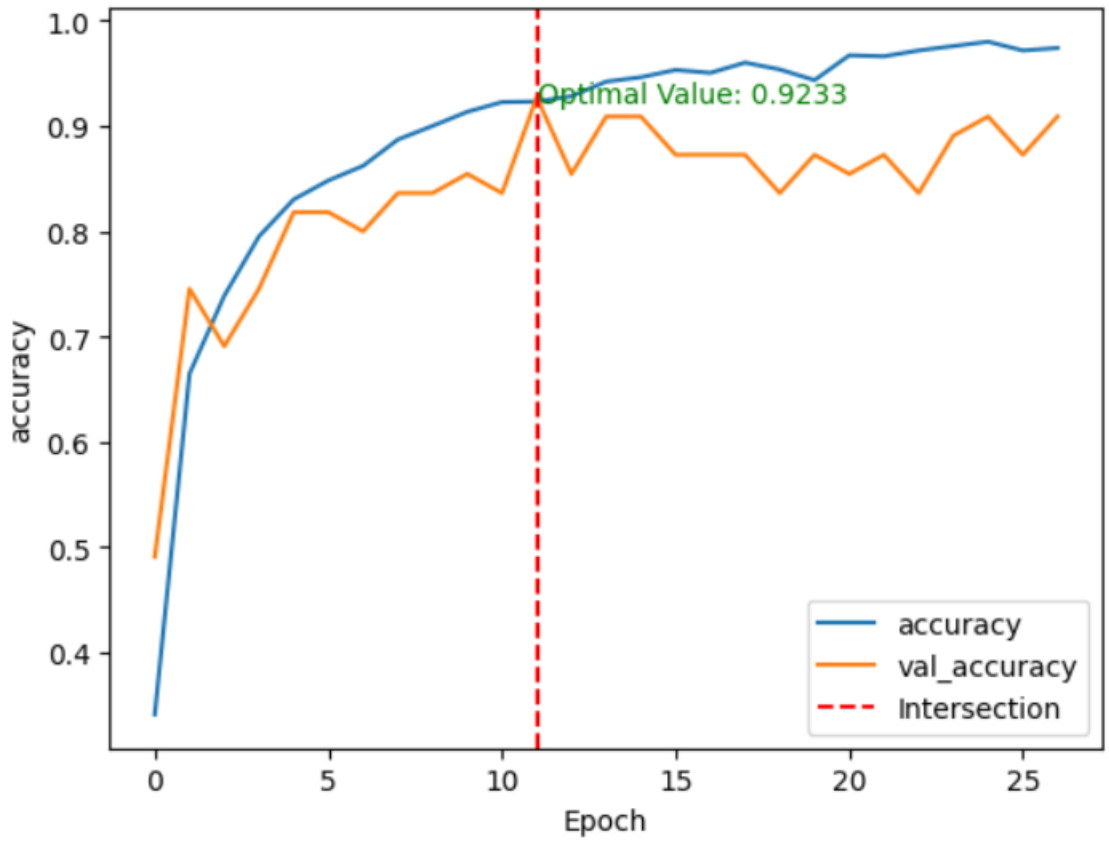

Figure 9: Performance plot after 30 epochs.

5.3 Confusion matrix

In order to understand how the algorithms works with prediction, the confusion matrix has been build. From the confusion matrix we can see that the prediction is mistaken (by just one element for each class) in label 2,6 and 9.

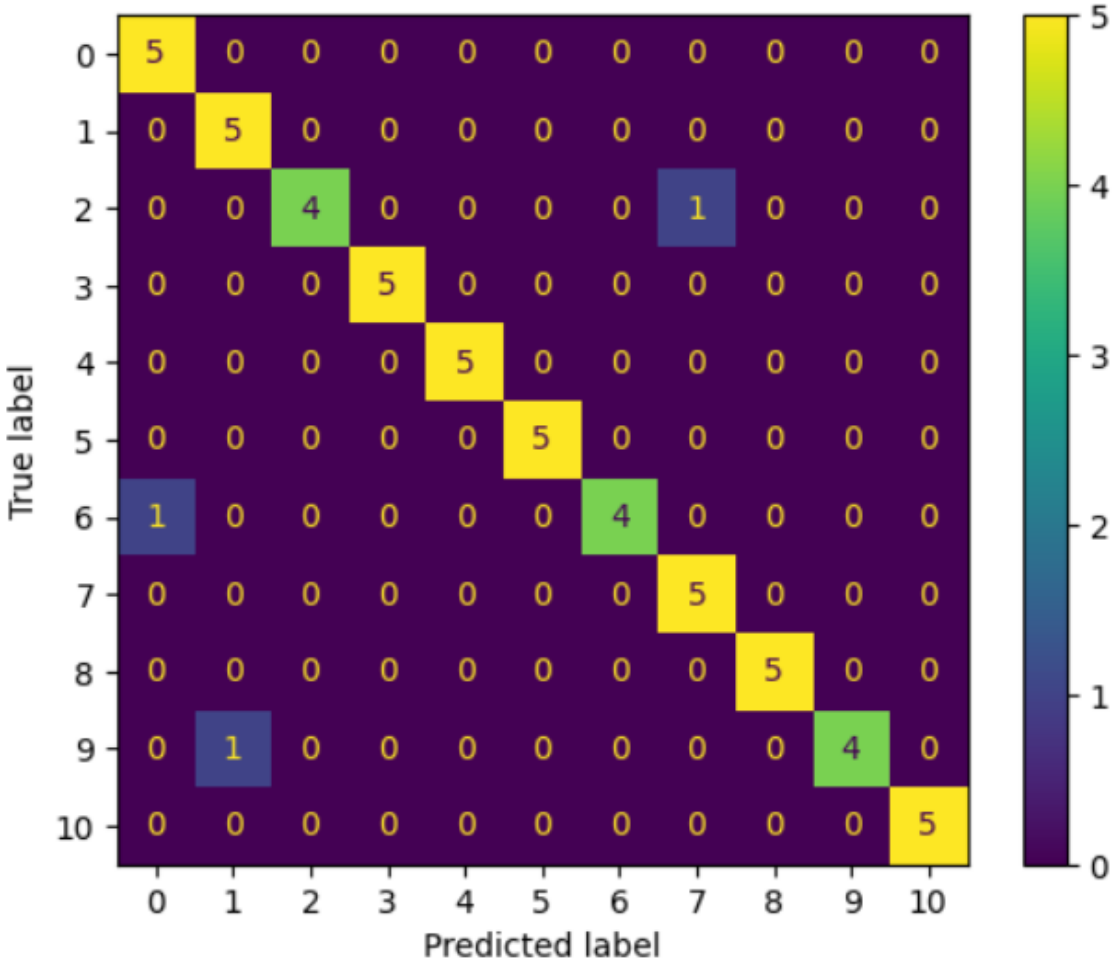

Figure 10: Confusion matrix.

6 Conclusions and further improvements

Overall it can be concluded that the CCN did a good job in predicting leaves with a 95% accuracy. The model that has been chosen to tackle this problem has been built taking into consideration the overfitting issue and regarding this in the training was also added an early stopping rule.
An improvement that could be done in this project is using cross validation instead of splitting the dataset into training, validation and test because the images that were in both validation and test were only 55, instead in the training we had more than 2000 images.
In addition to this improvement I would suggest to try a less complicated model to understand how it would have behaved with those data.

7 Reference

- Professor notes
- https://www.datacamp.com/tutorial/cnn-tensorflow-python
- https://www.datacamp.com/tutorial/convolutional-neural-networks-python
- https://medium.com/towards-data-science/dropout-in-neural-networks-47a162d621d9

8 declaration

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.