Plants leaves classifier using convolutional neural network

Giulia Franchi

March 20, 2023

Abstract

Convolutional Neural Networks (CNNs) are type of deep neural network algorithm very useful for processing image data, thus are widely used for tasks such as image classification. The aim of this work is to build a model using the architecture of convolutional neural network able to recognize and classify plants leaves into 12 classes of species.

1 Introduction

Neural network takes inspiration from the structure and function of biological neurons in the human brain. It is typically composed by layers, with each layer consisting of multiple nodes, the input receives input data and passes it on to hidden layers, which apply mathematical operations to transform the input and transfers it to the output layer, and then produces the final output to make predictions or decisions. CNNs are specialized type of neural network designed for processing images. They consist of a series of convolutional layers followed by pooling layers, connected to one or more fully connected layers. What makes interesting CNN rather then simple Neural Network is their ability to pick up or detect patterns and make sense of them.

For this purpose, I use a dataset from Kaggle with images of plants leaves belonging to 12 different class of species. Next sections I am going to explain how the dataset is composed and how I make the pre-processing, thereafter I will show the neural network that I created choosing the "best" hyperparameters. Indeed, when you build a model for neural network there are a lot of hyperparameters that need to be tuned, for this reason I use Optuna, a framework perfectly combines with PyTorch, to automate optimization of hyperparameters.

2 DATA AND METHODS

2.1 DATA

The dataset is taken by Kaggle and it contains 4503 images that have been classified among two classes i.e. healthy and diseased, but first the images are classified and labeled conferring to the plants. However, for the purpose of my project I decide to consider only the classification in different plants' species. The classes are 12 named: *Mango, Arjun, Alstonia Scholaris, Guava, Bael, Jamun, Jatropha, Pongamia Pinnata, Basil, Pomegranate, Lemon, and Chinar.*

The complete dataset was already divided into four distinct folders: one for training, one for validation, one for testing and one for predicting. I take three of them:

1. 'train' folder for training process, which contains 4274 images;
2. 'valid' folder for tuning the hyperparameters of the model, which contains 110 images;
3. 'test' folder for testing and predict labels once chosen the proper model, which contains 110 images.

Figure 1: An example of original image

2.2 Pre-processing

CNNs work with tensors which are matrices in three dimensions, so in addition to the width and the length, a tensor has got also the depth. First 2 dimensions are represented by the pixels of the image, while the depth represents its color. Considering these 3 different aspects I take some manipulation on image inputs:

- *Converting to grey scale.* In the source dataset the images are color, thus the value of depth is equal to three, this means that one image can be depicted by 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 (RGB), so the input in this case has got a very huge size. Converting them on grey scale changes the value of depth in 1, in this way the mathematical representation of an image is the classical two dimensional matrix.
- *Resizing.* The images in this dataset are in high resolution, 6000 x 4000 pixels, this implies a big amount of parameters. In this case it is very easy to overfit, at the same time estimating them is also very heavy from a computational point of view. So, I decide to decrease the format dividing the shape by a $resizefactor = 30$, so now images are 200x133 pixels.
- *Cropping.* I'm not still satisfied with the resulting format and I notice that there is some "useless" background, so I decide to remove a certain number of pixels from the edges. The amount of pixel removed is 400 divided by the $resizefactor = 30$, for both x-axis and y-axis. The result is an image cropped at the center with a size of 106x173.

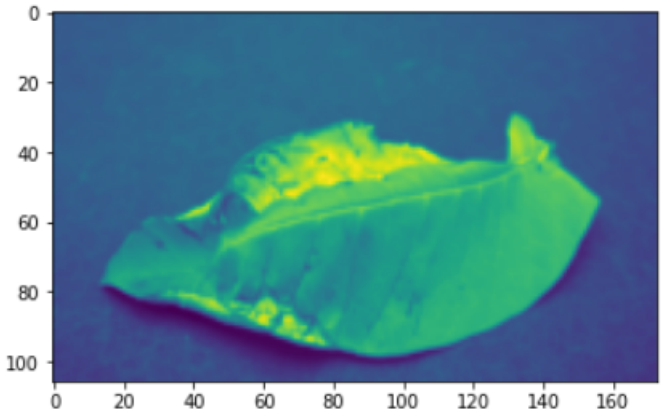

Figure 2: An example of image after manipulation

Before building the structure of my neural network model I do two more steps for preparing data:

- I obtain a new tensor that contains the numerical representation of the labels from the original list represented as strings. This numerical representation can be used as input to the algorithm, in fact it assigns an integer value from 0 to 11, each for one classes. In this case I don't do one-hot-encoding since in PyTorch library the $nn.CrossEntropyLoss$ uses the label encoded as integer number to index into the output probability vector to calculate the loss. This small but important detail makes computing the loss easier and is the equivalent operation to performing one-hot encoding, measuring the output loss per output neuron as every value in the output layer would be zero with the exception of the neuron indexed at the target class.

- The dataset retrieves features and labels one sample at time. While training model DataLoader, a PyTorch utility, loads samples in "minibatches" and reshuffle the data at every epoch to reduce model overfitting. I store $DataLoader$ object in a dictionary called $dataloaders$, using the mode name (train, valid and test) as the key. In this way each DataLoader contains images and labels corresponding to that mode and it will be helpful for training and testing models on different parts of the dataset. For my model I choose a batch size of 64 and I impose the shuffling in order to randomize samples in the dataset before being split into batches during each epoch.

2.3 Model for Convolutional Neural Network

CNNs have hidden layers called convolutional layers and also non-convolutional layers.

- Convolutional layers are able to detect patterns, they use a set of kernels, small matrix of weights with the function of filter, that are applied to the input image to extract features. These filters are actually what detects edges, circles, squares, corners, etc.,

- Max Pooling Layer, which is a kind of pooling layers, scans a region of the matrix and take the maximum value inside that region. Its scope is to reduce the spatial size of the feature maps, making the network more efficient.

- Activation functions are mathematical functions applied after each output layer. There are different types, for my model I use ReLu, the most common, that introduces non-linearity to the model and allowing it to learn more complex patterns in the data. Only after the final layer I apply Softmax function that takes values between 0 and 1, for this reason could be interpreted as probability distribution, giving the probability that an observation belongs to the classes.

- At the end of each structure there are dense layers fully connected with the inputs. They can be one or more and take the flattened output of the previous layer and maps it to the output of classes.

What I describe above is the fundamental structure of a convolutional neural network, in addition it's possible to use further techniques or hyperparameters to improve the performance of the model and to avoid overfitting:

- Batch normalization is a technique used for normalizing the inputs of each layer, both convolutional and fully connected, to have zero mean and unit variance across mini-batch samples. Thus, the distribution of the inputs to a layer changes over training process. I set the $Bias = False$ since I want the mean to be 0, so I don't want to add an offset (bias) that will deviate from 0.

- Dropout is a regularization technique that works by randomly dropping out, i.e. setting to zero, a fraction of the neurons in a layer during the training. This forces the network to learn more robust representations of the data by preventing the neurons from relying too heavily on any one input feature. In this case the rate will be tuned in the next section.

- Padding refers to the technique of adding extra rows and/or columns of zeros around the edges of an input image. Convolutional and pooling operations can reduce the spatial dimensions of the output, which can lead to a loss of important spatial information. In this case the value for padding is equal to 1 meaning that it will add a single row/column of zeros to each edge of the input feature map.

```python
def conv_block(in_c, out_c, k=3, pad=1):
    return nn.Sequential(
        nn.BatchNorm2d(in_c),
        nn.Conv2d(
            in_c,
            out_c,
            k,
            padding=pad,
            bias=False
        ),
        nn.MaxPool2d(2),
        nn.ReLU()
    )

def get_model(first_layer_kernel_size, dropout_rate):
  return nn.Sequential(
        conv_block(1, 16, first_layer_kernel_size),
        conv_block(16, 32),
        conv_block(32, 64),
        conv_block(64, 128),
        conv_block(128, 256),
        nn.Flatten(),
        nn.Linear(3840, 256),
        nn.ReLU(),
        nn.Dropout(dropout_rate),
        nn.Linear(256, 12),
        nn.Softmax(1)
  ).to(device)
```

Figure 3: General structure of my CNN

2.4 Hyperparameter Tuning and Model Selection

In this section I use *Optuna*, a brilliant tool for hyperparameter tuning. In fact, it uses a sampling algorithm for selecting the best hyperparameters combination from a list of all possible combinations. It concentrates on areas where hyperparameters are giving good results and ignores others resulting in time savings. This function is defined as below:

```python
def objective(trial):
    first_layer_kernel_size = trial.suggest_int('first_layer_kernel_size', 1, 5)
    lr = trial.suggest_float('lr', 0.0001, 0.001)
    dropout_rate = trial.suggest_float('dropout_rate', 0.1, 0.5)
    val_acc, _ = train(
      lr=lr,
      first_layer_kernel_size=first_layer_kernel_size,
      dropout_rate=dropout_rate,
      n_epochs=1
    )
    return val_acc

study = optuna.create_study(direction="maximize", sampler=optuna.samplers.TPESampler(seed=seed))
study.optimize(
    objective,
    n_trials=20
)
```

Figure 4: Optuna tuning hyperparameters

The three hyperparameters chosen are:

- *learning rate*, which is how deep or shallow we want to learn from the weight updates;
- *kernel size of the first convolutional layer,* which refers to the dimensions of "window" that is passed over the input feature map during convolutional operations in the first layer;
- *dropout rate* is a value between 0 and 1 and represents the probability of individual neurons being turned off during the training process. At the end I define the study object, in which what I want is to maximize the validation accuracy considering the first epoch, and then the number of trials, that is the number of times I want to repeat my study.

From this study it turns out that best parameters are:

- *kernel size*: 2
- *learning rate*: 0.00023208030173540176
- *dropout rate*: 0.13693543790751914

So, the final structure is summarized below

Layer (type)	Output Shape	Param #
BatchNorm2d-1	[-1, 1, 106, 173]	2
Conv2d-2	[-1, 16, 107, 174]	64
MaxPool2d-3	[-1, 16, 53, 87]	0
ReLU-4	[-1, 16, 53, 87]	0
BatchNorm2d-5	[-1, 16, 53, 87]	32
Conv2d-6	[-1, 32, 53, 87]	4,608
MaxPool2d-7	[-1, 32, 26, 43]	0
ReLU-8	[-1, 32, 26, 43]	0
BatchNorm2d-9	[-1, 32, 26, 43]	64
Conv2d-10	[-1, 64, 26, 43]	18,432
MaxPool2d-11	[-1, 64, 13, 21]	0
ReLU-12	[-1, 64, 13, 21]	0
BatchNorm2d-13	[-1, 64, 13, 21]	128
Conv2d-14	[-1, 128, 13, 21]	73,728
MaxPool2d-15	[-1, 128, 6, 10]	0
ReLU-16	[-1, 128, 6, 10]	0
BatchNorm2d-17	[-1, 128, 6, 10]	256
Conv2d-18	[-1, 256, 6, 10]	294,912
MaxPool2d-19	[-1, 256, 3, 5]	0
ReLU-20	[-1, 256, 3, 5]	0
Flatten-21	[-1, 3840]	0
Linear-22	[-1, 256]	983,296
ReLU-23	[-1, 256]	0
Dropout-24	[-1, 256]	0
Linear-25	[-1, 12]	3,084
Softmax-26	[-1, 12]	0

Total params: 1,378,606
Trainable params: 1,378,606
Non-trainable params: 0

Figure 5: Summary of my model

2.5 Results of the experiment

After selecting the best parameters, I run a model of CNN using the best parameters resulting from my validation set for 60 epochs.

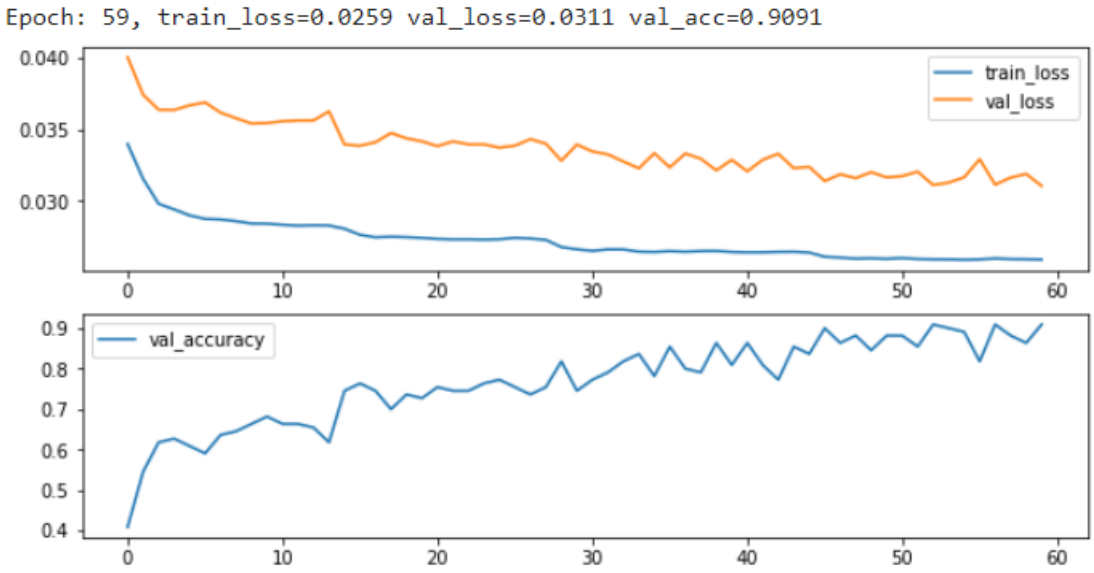

Figure 6: Performance of my model

Thanks to the graph is possible to see the evolution of the model during the training and validation process. The situation seems quite messy, the plots for validation loss and as well for the validation accuracy seem very jagged . This 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 in the previous epoch, since the batches are composed by a random draw with replacement from the training set. In addition the model seems to overfit a little bit, although the model is trained just only for 60 epochs, it might need more epoch in order to stabilize

Then, I evaluate the model on test set to see how it behaves on images never seen before. It turns out to output a test accuracy of 0.9091. Printing randomly 10 images belonging to test folder only one results misclassified:

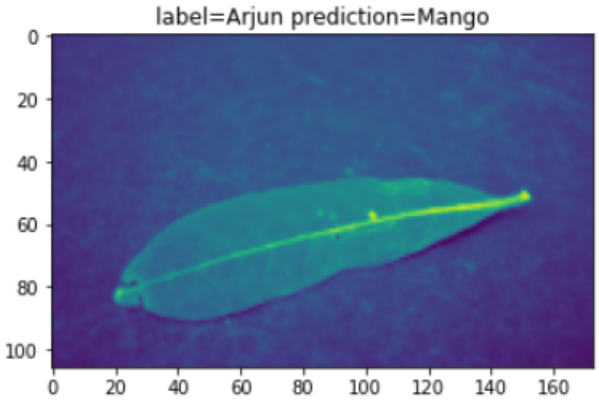

Figure 7: Image of test set misclassified

2.6 Conclusion

The CNN model developed for the plants leaves classification in this project could be surely further improved since it seems to overfit a little bit and it has the potential to obtain even an higher accuracy. There different techniques such as adding more data and retrain the algorithm on a bigger, richer and more diverse data. The size ca be artificially increased by applying data augmentation, a technique that can flip images, rotate, zoom out etc., this could be the model to generalize on different type of images. Moreover it's important to tune further hyperparamerts in order to improve the performance and save time for computation, for example finding the right size for batch of images and the right number of epochs is relevant to improve the general performance of my model.

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.

References

Lectures from course Algorithms for Massive Data 2022-2023

https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification