Convolutional Neural Network to classify leaves

Sara Paolini - 08400A

March 20, 2023

Abstract

This paper presents a model of Convolutional Neural Network (CNNs), a supervised machine learning technique, to classify pictures of leaves from various plant species. The dataset is obtained from Kaggle Plans_2. and was imported and mounted from Google Drive to the Colab runtime to easily access files and folders stored in the Drive from the Colab notebook. The images were processed using augmentation techniques to enhance the data and a Grid Search was performed to find the best hyperparameters and weights. The model was evaluated using various evaluation metrics. The analysis relied on TensorFlow, Keras, Layers, Sequential, and other libraries such as zipfile, pandas, numpy, and cv2 for handling the data and images.

1 Introduction

Convolutional neural networks (CNNs) is a type of NN architecture. A Neural networks (NNs) is an architecture inspired by the human brain's nervous system and the computational models consist of nodes (or "neurons") and edges (or "links") connecting them. Each neuron receives input from the neurons connected to its incoming edges, which are weighted and summed before being passed through an activation function to produce an output. The activation function transforms the output value of a neuron to a more useful shape for the neural network, such as limiting it to a binary value (Sigmoid function) or assigning a probability value (ReLu function). ReLu is the most used activation function and is the one I used in this project. Usally a bias is added before an activation function is to make the neurons activanting only if the weight sum. The learning process involves finding the right weights and biases.

The (CNNs) on the other side reduces the dimensionality of the image of the data set and keeps all the important features to be fed to the neural network for classification. This approach is necessary because using the original image would require expensive computation and time to process.

What is the architecture composed of? The mages are represented in RGB format so they are composed of three color channels - red, green, and blue - each represented by a two-dimensional matrix. A Kernel a filter used to extract features from an image and It's a matrix that moves over the input data (moving across the image left to right, top to bottom, with one-pixel column change on horizontal movements), performs the dot product with the sub region of input data and gets the outputs as the matrix of dot product. Stride is actually the amount of movement between application of the filter to the input image and is almost always symmetrical in the height and width dimensions. The output matrix is called feature map of the convoluted map. The next approach is the Padding, also called border problem solver because in the borders the filter is being applyed just one time and to solve this problem we put 0 padding on the border of the image. To reduce the dimensionality but preserve all the important features it's used a Pooling layers sampling features map. The two common pooling method are average and max pooling Once the pooled and padded feature is obtained the next step is to flatten it. It involves transforming the entire pooled feature map matrix into a single column which is then fed to the NN for processing.

2 Data and Preprocessing

In this section, I am going to illustrate the dataset that I use for this project and the pre-processing phase with train and test split that I run on the data for creating the Convolutional Neural Networks.

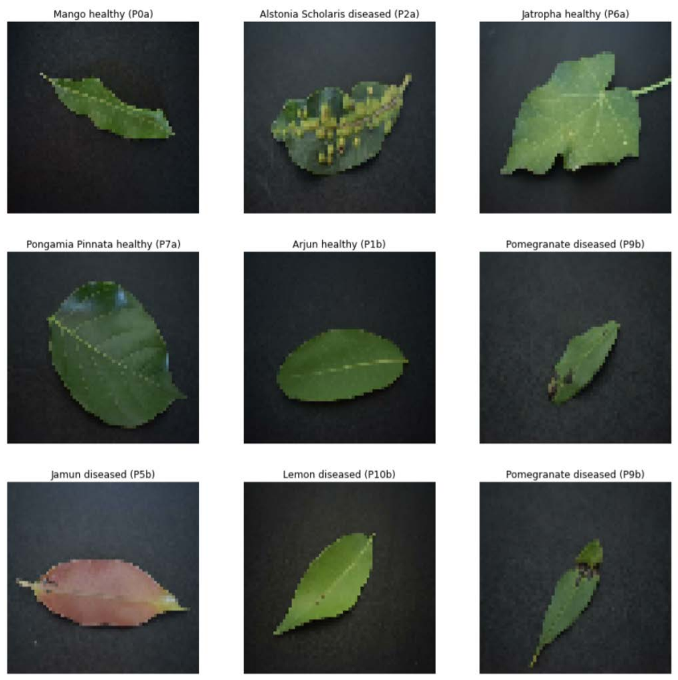

Mango healthy (P0a)
Alstonia Scholaris diseased (P2a)
Jatropha healthy (P6a)
Pongamia Pinnata healthy (P7a)
Arjun healthy (P1b)
Pomegranate diseased (P9b)
Jamun diseased (P5b)
Lemon diseased (P10b)
Pomegranate diseased (P9b)

Figure 1: Dataset as a graph of 9 images

2.1 Dataset

The dataset folder is called Plants_2 and is a collection of about 4503 images, with estension .JPG. Each file contains leaves for each plant species that the model will need to predict. The training file, test file, validating file contains 110 images each with 5 images for each type of plant represented in the dataset. The image to predictfile contains 8 images to try and predict for the purpose of testing your model.

I downloaded the dataset and unzipped it. Than I installed the software ImageMagick that is a part of Linux/Unix OS the that it's use to create, modify and visualized images, that I used for resize the dimension of the images to run the model in a more efficent way. This passage was necessary since the images were 1.73 megabyte each and I resized it to 11 kilobyte. Since I didn't want to download the data every time, after dowload the dataset and I load it in my Google drive folder.

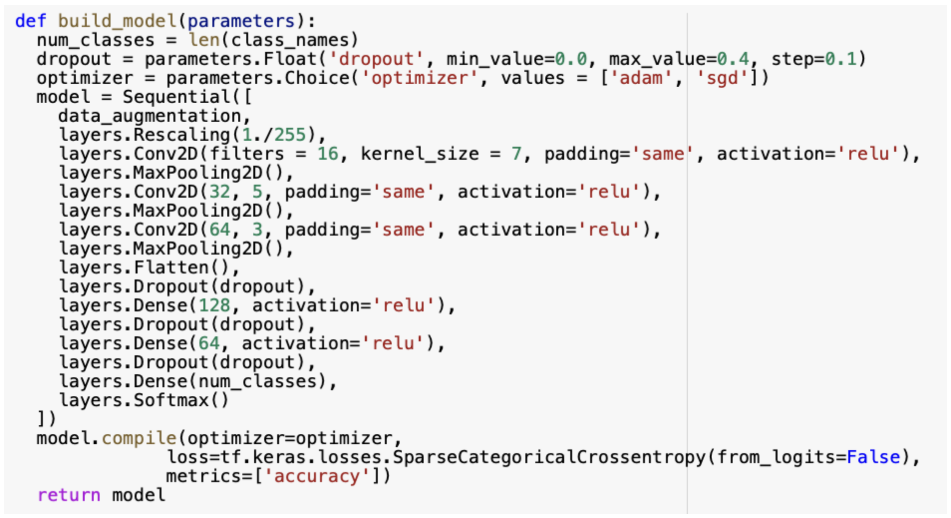
I plotted the dataset as a graph of 9 images (3 row and 3 colums) of dimension 15 x 15. They are sampled from the first batch of the training data set train_ds .

2.2 Autotune

I use three operations to load the data of the three dataset: caching, shuffling, and prefetching such that the Convolutional Neural Network will perform better and in the most efficent way. `Autotune`: Enables TensorFlow to choose the optimal number of parallel calls when processing the dataset. `Cache`: Caches the dataset in memory, which can speed up data processing by reducing the overhead of reading data from disk, so it allow me to memorized the the dataset in the memory (RAM) to avoid the need to reread the data. `shuffle`: Shuffles the dataset with a buffer size of 1000, which can help prevent overfitting during training by randomly presenting the data to the model. `prefetch(buffer_size=AUTOTUNE)`: Prefetches data for the next iteration while the current iteration is being processed, which can help reduce the time spent waiting for data and improve performance.

2.3 Training

I created the path of each subdirectory of the set of images of training, validation, test. I defined the following parameters: batch equal to 32 (which means that 32 images are processes at the same time in one iteration), height and width equal to 64 (in term of pixel). Those passages were done



```
def build_model(parameters):
  num_classes = len(class_names)
  dropout = parameters.Float('dropout', min_value=0.0, max_value=0.4, step=0.1)
  optimizer = parameters.Choice('optimizer', values = ['adam', 'sgd'])
  model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    layers.Conv2D(filters = 16, kernel_size = 7, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 5, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dropout(dropout),
    layers.Dense(128, activation='relu'),
    layers.Dropout(dropout),
    layers.Dense(64, activation='relu'),
    layers.Dropout(dropout),
    layers.Dense(num_classes),
    layers.Softmax()
  ])
  model.compile(optimizer=optimizer,
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])
  return model
```

Figure 2: Definition of the model.

to create the three dataset using image_dataset_from_directory, modul tf.keras.utils and the three subdirectory(train_path, val_path, test_path) of images that I will used to train, validate and test the neural network.

2.4 Data augmentation

In order to improve the model performances, I defined a data augmentation layer that will be used to transform the input images during training to improve the model's robustness and reduce overfitting. The model is defined as a sequence of image transformation layers:

1. RandomFlip "horizontal": it performs a horizontal random flip transformation on the image, inverting the image with respect to the horizontal axis.
2. RandomFlip "vertical": it performs a vertical random flip transformation on the image, inverting the image with respect to the vertical axis
3. RandomRotation: it performs a random translation of the image both horizontally and vertically, with a maximum translation of 5 percentage with respect to the image dimensions.
4. RandomZoom (0.1): it executes a random zoom on the image, with a maximum expansion of 10 pecent compared to the image dimensions.
5. RandomBrightness(0.2): it performs a random transformation of the brightness of the image, increasing or decreasing it by 20 percentage.
6. RandomContrast(0.2): it performs a random transformation of the image contrast, increasing or decreasing it by 20 percentage.

3 Model

To built the convolutional neural network (CNN) model I used the TensorFlow framework. The "parameters" parameter indicates the user choice for some model hyperparameters, such as dropout (from a miminum of 0 to a maximum of 0.4) and optimizer (that could be either adam or sgd) . The learning rate it's 0.01 on default. The model scales the input pixel values from [0, 255] to [0, 1] . Each Conv2D layer (so a single computation unit that processes the input data and produces output data) applies a convolution operation on the input image using a set of filters to extract features. The first Conv2D layer has 16 filters of size 7x7, the second layer has 32 filters of size 5x5, and the third layer has 64 filters of size 3x3. MaxPooling2D() layer downsamples the output of the convolutional layer by taking the maximum value in a pool of size 2x2. Figure 2 and Figure 3

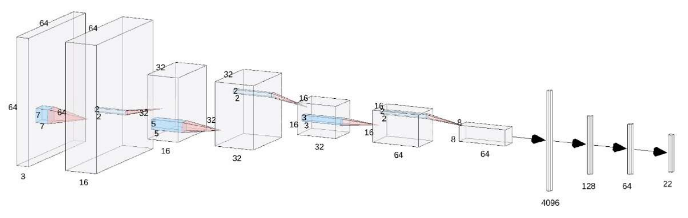


Figure 3: Architecture of the model. NN_SVG.

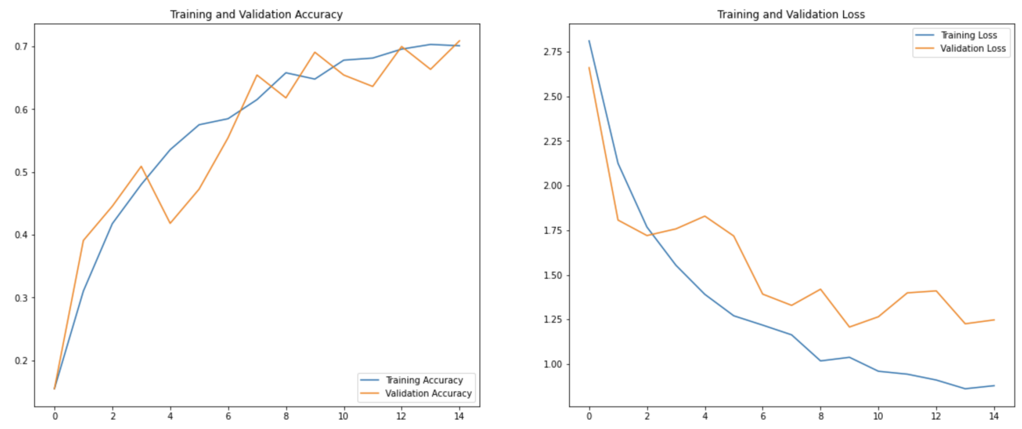


Figure 4: Evolution of the accuracy and the loss.

I used Keras Tuner library that to pick the optimal set of hyperparameters for the model. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter tuning or hypertuning.

Before fitting the model I created an Early stop to monitor the validation loss. The epoch from which to start monitoring is set to 0, so monitoring starts from the beginning, and the model stops if the validation does not improve for 5 epochs. The weights of the model will be restored to the epoch with the lowest validation loss.

Then I initialized a RandomSearch (a family of numerical optimization methods that do not require the gradient of the problem to be optimized) to perform the hyperparameter search. It will evaluate the performance of each model that it tests during the search and the tuner will be looking for models that perform well on the validation accuracy metric.

3.1 Loss and accuracy curves

Then I trains a TensorFlow Keras model using the fit() method and I will returns a history object that contains the training and validation loss and accuracy values for each epoch. The aim is to plot the learning curves and analyze the performance of the model during training.

The plot represented in Figure 4 and it is possible to see that the model doesn't have any problem of overfitting. From epoch 4 on, the validation accuracy stops increasing and decreases drastically but then goes back to the level of the training loss and stays constant through the epochs.

4 Evaluation metrics

In this part I'm gonna talk about commonly used evaluation metrics in machine learning that measure different aspects of the model performance. To do that it was necessary to split between true labels and predicted label. The predicted label is obtained by taking the argmax of the output of the predict() method and because the model outputs probabilities, it will select the class with the highest predicted probability.

All the functions listed below takes two or three arguments:

1. Trues: lists of true labels.
2. Preds: lists predicted labels.
3. Average: it specifies the type of averaging to use when computing precision. In this case set the average to 'macro', so the precision is computed as the unweighted mean of precision for each class.

   The true positives are the number of positive instances that the model correctly predicted as positive. False negatives are the number of positive instances that the model incorrectly predicted as negative

4.1 Accuracy score

It is a measure of how often the model predicts the correct label out of all the instances it predicted. It is calculated as the ratio of the number of correctly predicted instances to the total number of instances.

`Accuracy result`: 0.7454545454545455

4.2 Precision score

It is a measure of how often the model correctly predicts a positive instance (i.e., true positive) out of all the instances it predicted as positive (both true positives and false positives). It is calculated as the ratio of the number of true positives to the total number of predicted positives.

`Precision`: 0.7406926406926408

4.3 Recall score

It is a measures of the ability of the model to correctly identify all positive instances out of all actual positive instances in the data.

Recall $= \frac{TruePositive}{TruePositive+FalseNegatives}$

`recall`: 0.7454545454545454

4.4 F1 score

It combines both precision and recall to provide an overall measure of the model's ability to correctly classify positive and negative instances. F1 score ranges from 0 to 1, with a value of 1 indicating perfect precision and recall, and a value of 0 indicating that the model fails to correctly classify any of the positive instances.

F1 score $= 2\frac{precision*recall}{precision+recall}$

`F1 score`: 0.7164994096812278

5 Conclusion

The CNN model developed for classify pictures of leaves has the potential to obtain even an higher accuracy on the validation set. Of course, I could have tried different architecture but the model doesn't have a bad result for a CNN with only two hyperparameters tuned.

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