Diseased Leaves Image Detection using Convolutional Neural Network (CNN)

Jose German, Mohammed Araf Mir,
School of Continuing Studies, York University, Machine Learning Unit, Toronto, Ontario, Canada
jose.german@outlook.com, miraraf@gmail.com

ABSTRACT

In this project, we address the problem of binary classification for (plant) leaf images to determine whether a given leaf is diseased or healthy. Leveraging TensorFlow, a powerful deep learning framework, we designed and implemented a convolutional neural network (CNN) to perform image classification. The primary dataset consists of high-resolution images of plant leaves(taken from Kaggle.com), categorized into two classes: diseased and healthy. Our approach involves several key steps: data preprocessing to enhance image quality and normalize inputs, augmentation techniques to increase dataset variability and improve model robustness, and the design of a CNN architecture to capture the detailed features of leaf images. The trained model demonstrates high accuracy in distinguishing between diseased and healthy leaves, offering a practical solution for automated plant disease detection. Key findings and future work include the potential for expanding the model to recognize specific diseases, integrating the system into mobile applications for real-time field use, and exploring additional learning to adapt the model to different plant species and diseases.

Keywords: Image Processing , Classification, TensorFlow, Keras, Convolutional Neural Network, Normalization, Optimizer, Accuracy, Validation, Loss, augmentation, Thresholding

I. INTRODUCTION

The main purpose of this image classification project is to diagnose whether images of tree leaves are healthy or diseased. This project posed an interesting problem in which not all tree leaves are the same and also the diseases impacting the leaves are also different by nature. Our focus was to train a Convolutional Neural Network (CNN) model to focus and detect on the leaves themselves and whether they contained a disease.

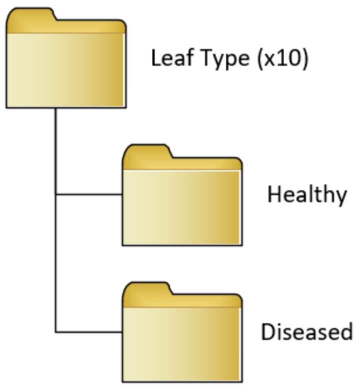
Our particular dataset contained 4,232 images in total. The dataset was divided by 10 tree types and then by healthy versus diseased. All images were 4000x6000 pixels in dimension, which translated to great overall image quality and no further preprocessing was required to handle different image dimensions. Class balance between healthy and diseased could be improved by augmenting the original dataset with further images.

II. METHODS AND MATERIAL

A. Dataset

This dataset has been taken from kaggle.com. It consists of 4235 images across 10 different leaf types.

Basic folder structure:



B. Data Preprocessing

(a) **Data Subset:** 4235 number of leaf images were there in the original dataset and size of this dataset is 6.44 GB. We have decided to use only a subset of these images for the training & validation purpose to reduce the processing time and to reduce the model complexity. A new program was developed that will split the image dataset (both for *healthy* and *diseased* images) and take only a fraction of it based on a threshold value decided by the user. In this example, we have used a threshold as 0.3 i.e only 30% of the images were taken from each of these categories to make a data subset. Please refer to section IX. References (subsection 3a) for the code snippet. Similarly, we have developed a program that will take a certain fraction of images from each type of the images and then combine them into either a healthy or diseased category. This is how the overall size of the image subdataset has been reduced to 1.9GB (from 6.4 GB) that helped us to train & validate the model faster /cheaper without compromising the performance and accuracy much.

(b) Image Thresholding: Our images were very good quality and they all contained a clear image of each leaf. However, all leaf images had a dark textured background which in model training created an overfitting condition. To deal with this hurdle we considered a couple of different methods to deal with the background of images. One method was to completely remove the background and make it transparent but we could not find enough research to validate if this method would prove effective in model training or how the model would interpret transparent pixels.
The other method and it is the one we decided to use is called image thresholding.

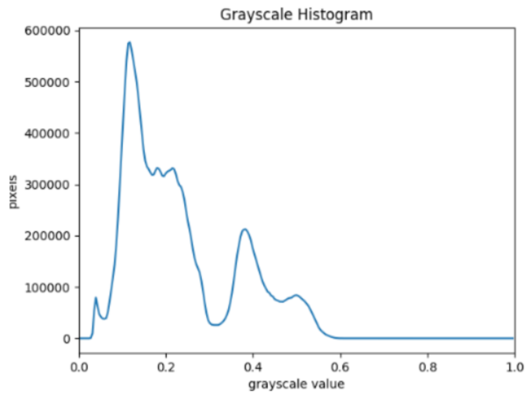
(Original image)

Thresholding is a method or type of image segmentation, where we first convert the image into grayscale or black and white to isolate areas of interest. In our case the area of interest is the leaf and isolating it from the background. Once isolated we can treat the background and make all pixels zero or black.

In the next image we can see that we have converted the original image into grayscale. Also important to mention that we applied a Gaussian filter to slightly blur the image and eliminate any unwanted pixels.

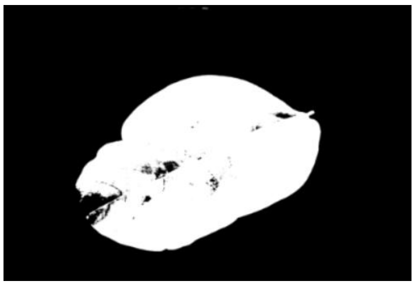
(Image converted to grayscale)

After the image is converted we generate a histogram of the image.

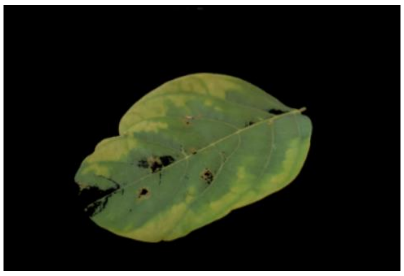


(Histogram of grayscale image)

In the histogram we can observe some peaks of interest. The large one which has a grayscale value of around 0.15 and the other peak around 0.4. The pixels closer to 0 (zero) are darker which in our case is the background. The smaller peak is the pixels related to the leaf. The point between the two prominent peaks is around 0.33 which basically isolates the leaf from the background.Once we know how to isolate we can create a mask based on the threshold.


(Binary mask of image)

The mask is then used to select the interesting part of the image or in our case the leaf. All other pixels are turned off or set to zero (0) or black.


(Image with background removed)

Above you can see the final image with the background removed using the Thresholding method.
The downside of the simple thresholding technique is that we have to make educated guesses about where to isolate based on the threshold value we picked. To overcome this hurdle we used Otsu's method which determines the threshold value automatically.

(c) Random image augmentation: To deal with the imbalance and introduce more variety of images to our model we employed the use of random image augmentation. This prevented the model from overfitting on the similarity of the images.
This method takes images and randomly applies one of four changes: rotation, translation, flip and contrast.

C. Framework, Libraries & Packages:

(a) Colaboratory:
Google's Colaboratory has been used as a platform. Colab is a hosted Jupyter Notebook service that requires minimal setup to use and provides free of charge access to computing resources. For our model training we procured additional compute units and trained using T4 GPU runtime.

(b) Framework:
TensorFlow has been used as a machine learning framework. TensorFlow version 2.15.0 has been used in this project

(c) Packages & Libraries:
keras,matplotlib, PIL, tensorflow_datasets, numpy, scikit-image (thresholding)

(d) Other Tools/Techniques:
Standard python libraries like matplotlib, numpy, pandas, re are also used in this project

III. Algorithm

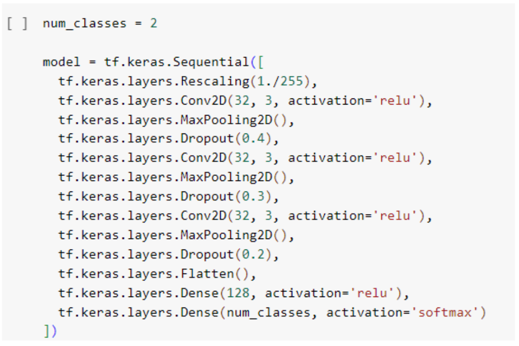
(a) **Label Generation:**There are 10 different types of leaves present in the given dataset. (we can add the list as an appendix). For each of the leaf types, it is further categorized into two groups: one for healthy leaves and other is for diseased leaves. Since, the objective of the project is to identify whether a given image is healthy or diseased i.e this is considered as a binary classification problem. This is the reason we have developed a program that will copy all the healthy leaves and put them together into a folder . Each of these images are labelled as "*healthy*" images. Similarly, we have coped all the diseased leaves programmatically and put them together into a different folder . Each of these images are labeled as "*diseased*". Please refer to appendix section XYZ for the python code that was developed to do this work.

(b) **CNN Model:**
A neural network in which at least one layer is a convolutional layer. A typical convolutional neural network consists of some combination of the following layers:
(i)convolutional layers
(ii)pooling layers
(iii)dense layers

IV. Experiment

(a) Baseline Model: It is important to establish a baseline model in order to know what is truly working and not working as you work through changes. In our particular case, we made lots of changes to the model by adding layers, changing the number of filters, changing the scaling factor, changing dropout layers, etc. As we progressed through changes we noticed our performance was not getting any better. This is when we knew we had to look at more than just the model and we started focusing on the images.

Code snippet:


```
[ ]  num_classes = 2

     model = tf.keras.Sequential([
       tf.keras.layers.Rescaling(1./255),
       tf.keras.layers.Conv2D(32, 3, activation='relu'),
       tf.keras.layers.MaxPooling2D(),
       tf.keras.layers.Dropout(0.4),
       tf.keras.layers.Conv2D(32, 3, activation='relu'),
       tf.keras.layers.MaxPooling2D(),
       tf.keras.layers.Dropout(0.3),
       tf.keras.layers.Conv2D(32, 3, activation='relu'),
       tf.keras.layers.MaxPooling2D(),
       tf.keras.layers.Dropout(0.2),
       tf.keras.layers.Flatten(),
       tf.keras.layers.Dense(128, activation='relu'),
       tf.keras.layers.Dense(num_classes, activation='softmax')
     ])
```

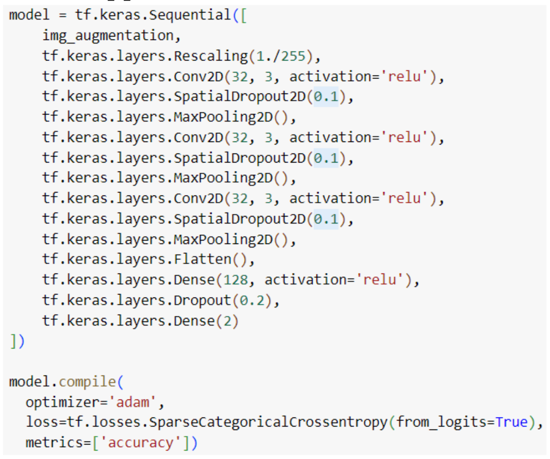
Baseline Model Train Accuracy: 96%
Baseline Model Validation Accuracy: 82%
# images used to train the model: 1007
# images used to validate the model: 251

Please refer to section IX. References (subsection 3b) for the baseline code snippet (written in python)

(b) Final Model:

Our final model consists of four convolutional layers all having the same number of filters and 'relu' activation. To help with overfitting we use a dropout layer set to 0.2.

Code snippet:



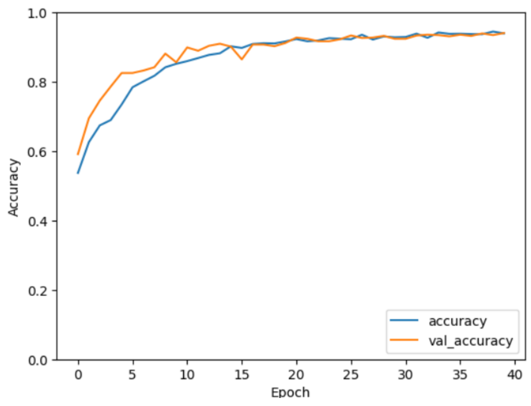
```
model = tf.keras.Sequential([
    img_augmentation,
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.SpatialDropout2D(0.1),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.SpatialDropout2D(0.1),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.SpatialDropout2D(0.1),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(2)
])

model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

Final Model Train Accuracy: 94%
Final Model Validation Accuracy:94%
# images used to train the model: 3390
# images used to validate the model: 842

(c) Model Accuracy:

Model accuracy for training and validation.



Final test accuracy for mode after 40 epochs:

```
Test accuracy: 0.940617561340332
```

(d) Model Loss: The CNN model was trained by having 20 epochs and 32 images were processed in each of the epochs. From the chart below, we could see that both model training and validation losses were reduced gradually as the number of iterations increased. Final model training loss was 0.11 and model validation loss was 0.49. We could have minimized training losses further by adding more training data or by increasing the number of epochs. However, the model validation loss could not not be improved further simply by adding more training or validation data.

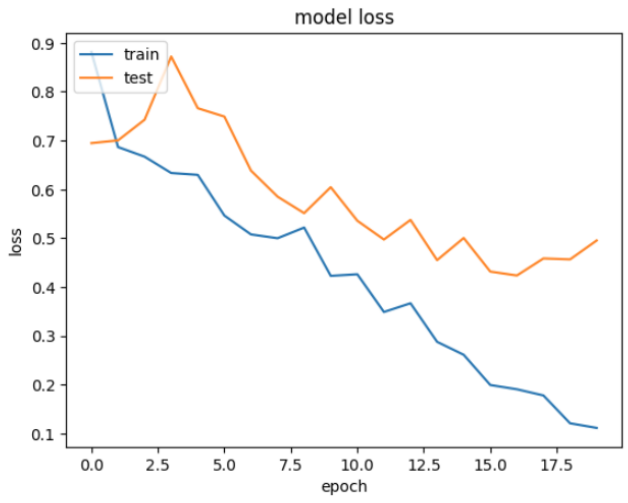
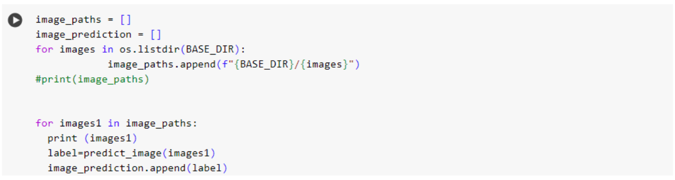


Figure 1: Model Training and Testing Loss by Epoch

V. RESULTS AND DISCUSSION

We have chosen 7 images that are unseen by the model for predicting purposes. The model was able to accurately predict the right classification for all cases with confidence level between 0.60 and 0.99. We have verified the classification predicted by the model by visualizing the image to ensure its accuracy and we were satisfied with the results. We can see this CNN model demonstrated strong performance with high accuracy and low loss across training, validation, and testing datasets.

Code Snippet:



```
image_paths = []
image_prediction = []
for images in os.listdir(BASE_DIR):
            image_paths.append(f"{BASE_DIR}/{images}")
#print(image_paths)


for images1 in image_paths:
  print (images1)
  label=predict_image(images1)
  image_prediction.append(label)
```

Result/Output:

The probability is **0.97** and the predicted label for this image is **diseased**
Image full path is:
/content/drive/MyDrive/YU-ML-Proj-1/CSML-1020/Group_Project/CNN-Araf/image_validate//0022_0089.JPG

Please refer to section IX. References (subsection 3b) for complete result .

VI. Future Scope:

(a) This model can be deployed in the Cloud Platforms (like Google Cloud Platform (GCP or Microsoft's Azure) to

leverage the full potential of Cloud architecture so that it can be used at scale.
(b) The Model can be trained using entire dataset and that could further improve the testing and validation accuracy
(c) Additional image processing techniques can be applied to remove the noise present in the image datasets.

VII. CONCLUSION

In conclusion, the binary image classification project was challenging from the start. Typically, the focus at start becomes the model and what changes can be made in order to improve its performance. This is still the case but it became apparent that our focus needed to be on the images. Similar to when you work on other machine learning projects, the exploratory data analysis phase is one where you have the opportunity to learn from the data. You determine the important things like missing values, how balanced are your classes, and many more key information about your dataset. In image classification the EDA phase really becomes about learning about the images. What do the images look like, how is the quality of the images, are the classes well balanced and more importantly are there any elements in the image which may become detrimental when training the model. In our particular case, we noted the background of the image might cause issues when training. Although it was dark and very dissimilar to the important part "the leaf", its texture posed a challenge when training the model. The model quickly treated the texture as data and became overfitted on its information. Implementing the typical code changes to deal with overfitting did not have much impact on the results.

There are quite a few methods that can be employed to manipulate images but no methods have been proven to work with machine learning models and the impacts to their performance. This makes deciding on what steps to take to modify images a difficult one. We decided to take a more conservative approach and use methods that have been proven in other machine learning projects. Thresholding and image augmentation quickly proved to us that paying attention to the images is very important to improve performance of machine learning models.

VIII. Acknowledgments

We would like to express our sincere thanks & profound gratitude to our professor *Mr. Harish Bhaskar* for his expert coaching, advise to make this project successful.

We would also like to acknowledge our colleagues *Mr. Luiz Tavares* and *Mr. Abdela Mossa* for their ideas and support in the project.

Our sincere thanks to York University's project Coordinator and staff members for their support.

IX. REFERENCES

[1] Healthy vs. Diseased Leaf Image Dataset: https://www.kaggle.com/datasets/amandam1/healthy-vs-diseased-leaf-image-dataset

[2] Loading and preprocessing images using TensorFlow: https://www.tensorflow.org/tutorials/load_data/images

[3] Project Code Repo:
https://github.com/miraraf2023/ML1030

a. Code to split the original image dataset & label it. https://github.com/miraraf2023/ML1030/blob/main/Course-3ML-project2.ipynb

b. The Code for baseline CNN Model: https://github.com/miraraf2023/ML1030/blob/main/wip_image_process_leaf_tf.ipynb

c. The final Model Code:
https://github.com/miraraf2023/ML1030/blob/main/final_leaves_v2_wip.ipynb

d. Additional CNN project code
https://github.com/josegerman/Machine-Learning/blob/570ecc35cd197fbb600cb897a9b518ae5dadbfd3/cnn-leaf-class_v2.ipynb

[4] Otsu's method:
https://en.wikipedia.org/wiki/Otsu%27s_method