Final Report: Plant Leaf Diseases Diagnosis with Deep Learning

Why this project?

My father studied plant pathology in college back in the 1980s. Growing up, his field trip stories to examine different kinds of plants and crops were my favorite. From his narratives, diagnosing plant diseases and recommending precise treatments demands a great deal of expertise and effort. What if the required expertise is missing, such as when farmers plant something new? Or home gardeners and houseplant hobbyists who just started their planting journey? A machine learning based automatic plant leaf diseases identification tool becomes very helpful. It can bring the required expertise to everyone as long as you have access to the tool.

I have a firsthand experience where I helped my mother treat her sick indoor plants. Unaware of the exact disease, my mother used several pesticides, the strong smell of which made everyone flee the house. It wasn't until we got advice from an in-door plant expert (a college peer of my father), who suggested an appropriate treatment, that the flower began to recuperate. How I wish I had a mobile app that could identify plant diseases by simply taking a picture of the sick plant.

Such a deep learning based plant disease identification model holds significant promise for modern automated agricultural systems. It can also be further transformed into a mobile app benefiting any individuals who love planting at home. Indeed, there are existing apps like Plant Doctor, Plantix, Agrio, and several others that serve similar purposes.

Data

In order to build a machine learning model for plant disease diagnosis, images of plant leaves with different diseases are required. After some search online, I found the following dataset[1] on Mendeley Data which was originally from a published journal paper[2].

The data consists of 61,486 images from 15 species including common fruits and vegetables and one background class with images that are not plants. Not only images of sick leaves are included but also images of healthy leaves.

Here is the list for the 39 classes in the data.

1.**Apple**_scab, 2.Apple_black_rot, 3.Apple_cedar_apple_rust, 4.Apple_healthy,

5.**Background**_without_leaves, 6.**Blueberry**_healthy, 7.**Cherry**_powdery_mildew,

8.Cherry_healthy, 9.**Corn**_gray_leaf_spot, 10.Corn_common_rust,

11.Corn_northern_leaf_blight, 12.Corn_healthy, 13.**Grape**_black_rot,

14.Grape_black_measles, 15.Grape_leaf_blight, 16.Grape_healthy,

17.**Orange**_haunglongbing, 18.**Peach**_bacterial_spot, 19.Peach_healthy,

20.**Pepper**_bacterial_spot, 21.Pepper_healthy, 22.**Potato**_early_blight,

23.Potato_healthy, 24.Potato_late_blight, 25.**Raspberry**_healthy, 26.**Soybean**_healthy,

27.**Squash**_powdery_mildew, 28.**Strawberry**_healthy, 29.Strawberry_leaf_scorch,

30.**Tomato**_bacterial_spot, 31.Tomato_early_blight, 32.Tomato_healthy,

33.Tomato_late_blight, 34.Tomato_leaf_mold, 35.Tomato_septoria_leaf_spot,

36.Tomato_spider_mites_two-spotted_spider_mite, 37.Tomato_target_spot,

38.Tomato_mosaic_virus, 39.Tomato_yellow_leaf_curl_virus

[1] J, Arun Pandian; G., Geetharamani (2019), "Data for: Identification of Plant Leaf Diseases Using a 9-layer Deep Convolutional Neural Network", Mendeley Data, V1, doi: 10.17632/tywbtsjrjv.1

[2] Geetharamani G., Arun Pandian J., "Identification of plant leaf diseases using a nine-layer deep convolutional neural network", Computers & Electrical Engineering, V76, p323-338, June 2019

Method

The task is to build a supervised classification model for images. This is a fundamental task in computer vision and several models have been developed for this purpose:

1. **Convolutional Neural Networks**: CNNs are the backbone of most modern image classification techniques. They can automatically learn the spatial features from images.
2. **CNN-based Models**: ResNet (Residual Networks) developed by Microsoft, GoogLeNet introduced by Google, VGGNet developed by Visual Graphics Group at Oxford and many more.
3. **Transformer-based Models for Vision**: Originally designed for natural language processing, transformers have been adapted to vision tasks with promising results.

In this work, a 9-layer CNN model was selected for the plant leaf disease diagnosis task inspired by the original journal paper. CNN models are the foundation of more advanced models and they are well developed and widely used in the deep learning community.

Here is the original CNN model structure used in this work for preliminary model evaluations:

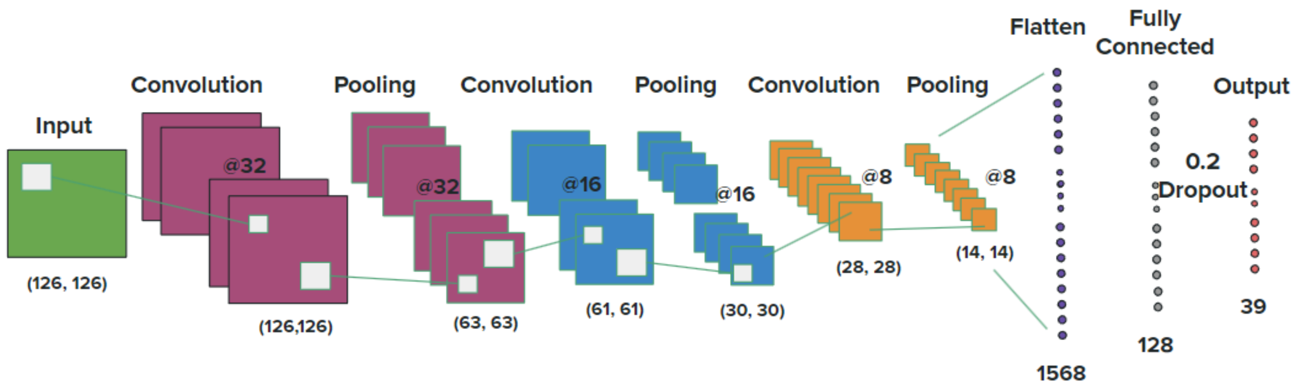


Figure 1a. Structure of the original 9-layer CNN model used in this work.

After hyper-parameter tuning, here is the tuned CNN model structure with better accuracy:

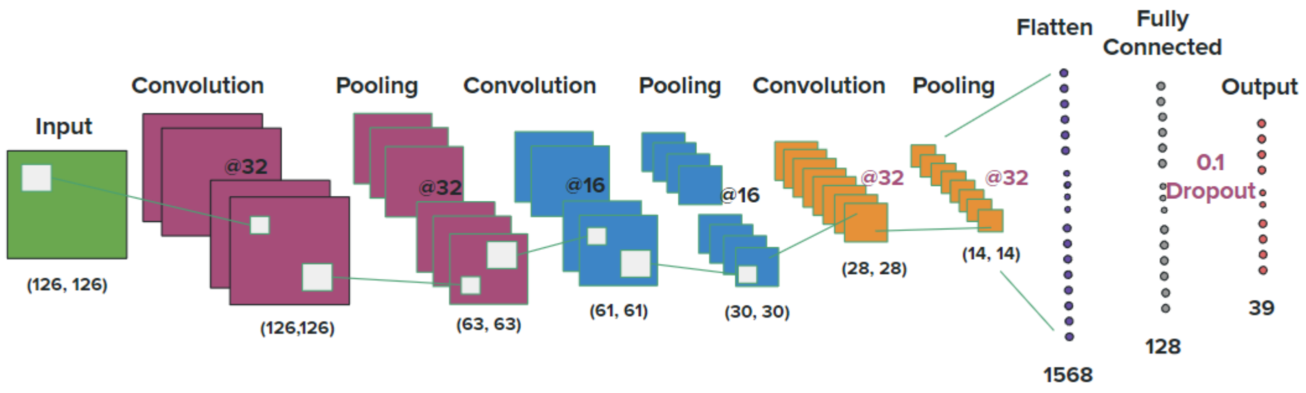


Figure 1b. Structure of the tuned 9-layer CNN model used in this work.

Data Challenges

After wrangling the raw image data, there are 39 classes of images as described by the data source. Number of images per class after the train/test split (test size = 0.2) is plotted in Figure 2.

Challenge 1: Insufficient Data

From Figure 2, there are several classes with a very limited number of images, the smallest number being only 122. With such insufficient data, it is hard for any model to give accurate predictions.

The ideal solution will be collecting more data for the classes with limited amounts of samples. In this work, inspired by the original journal paper, **data augmentation** using existing data was tested to increase the number of samples.

Challenge 2: Class Imbalance

Sample sizes of different classes vary from 122 to 5000+. Class imbalance is a challenge in the raw data. By carefully selecting model metrics, the imbalanced testing data may not be a big issue. However, the imbalanced training data certainly will have negative impacts on model building.

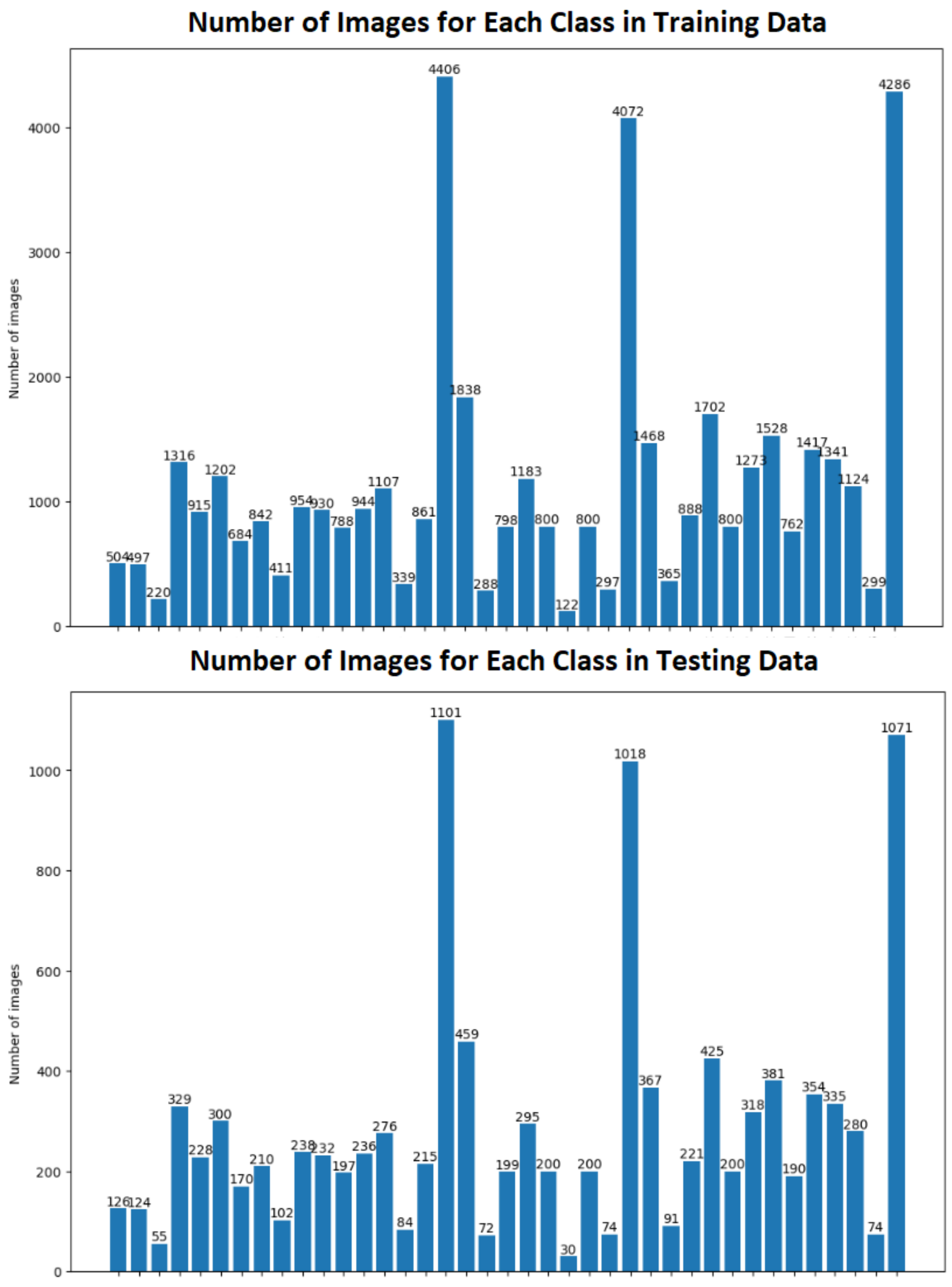


Figure 2. Sample size for each class in training (upper) and testing data (lower).

Common strategies to address class imbalance are resampling the data by upsampling or downsampling, data ensembling (**Bootstrapping**), algorithm modifications and model ensembling (**Bagging**). The best practice will be combining multiple methods.

In this work the following approaches in Table 1 were tested:

Approaches	Data Imbalanced	Upsampling (Data Augmentation)	Data Ensembling (Bootstrapping)	Model Ensembling (Bagging)
1	Yes	No	No	No
2	Yes	Yes	No	No
3	No	No	Yes	Yes
4	No	Yes	Yes	Yes

Table 1. Different approaches tested to address data challenges

Data Augmentation

Figure 3 shows 3 randomly selected images in the raw data with their corresponding labels. As mentioned in the previous section, data augmentation technique can be helpful to increase sample size using existing images.

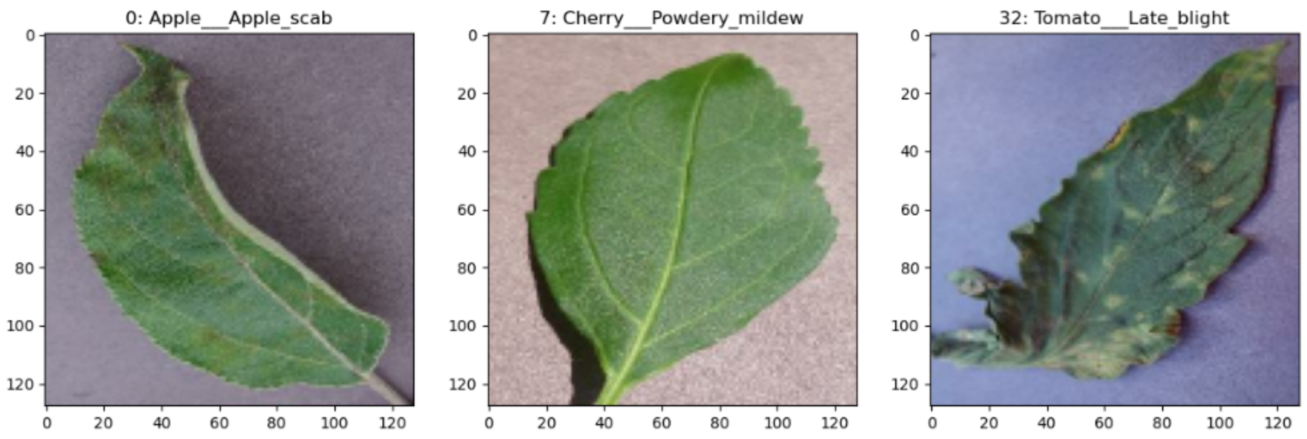
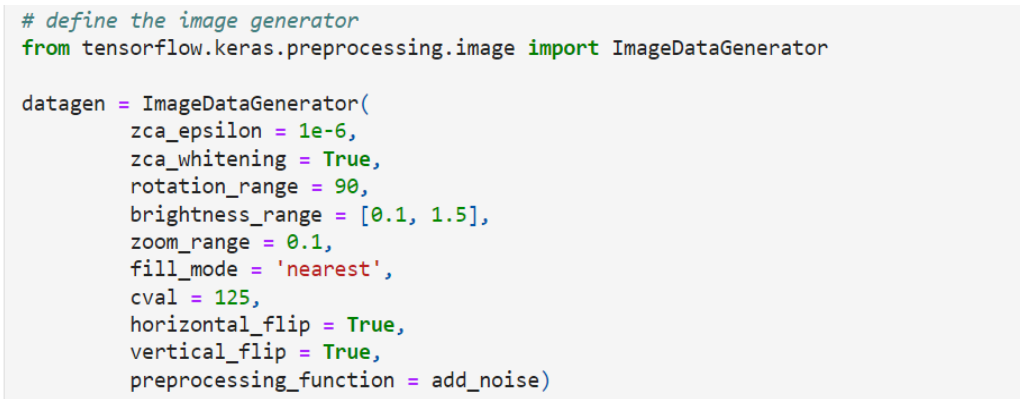


Figure 3. Random images from raw data with labels.

Tensorflow.Keras.PreprocessingImage.ImageDataGenerator was utilized in this work to perform data augmentation by rotating, flipping, scaling, zooming and whitening the original images. A user-defined function adding random noise to images was added to ImageDataGenerator. Data augmentation was performed by batches to increase sample size to at least 900 for each image class. Figure 4 shows some images after augmentation.



```python
# define the image generator
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
         zca_epsilon = 1e-6,
         zca_whitening = True,
         rotation_range = 90,
         brightness_range = [0.1, 1.5],
         zoom_range = 0.1,
         fill_mode = 'nearest',
         cval = 125,
         horizontal_flip = True,
         vertical_flip = True,
         preprocessing_function = add_noise)
```

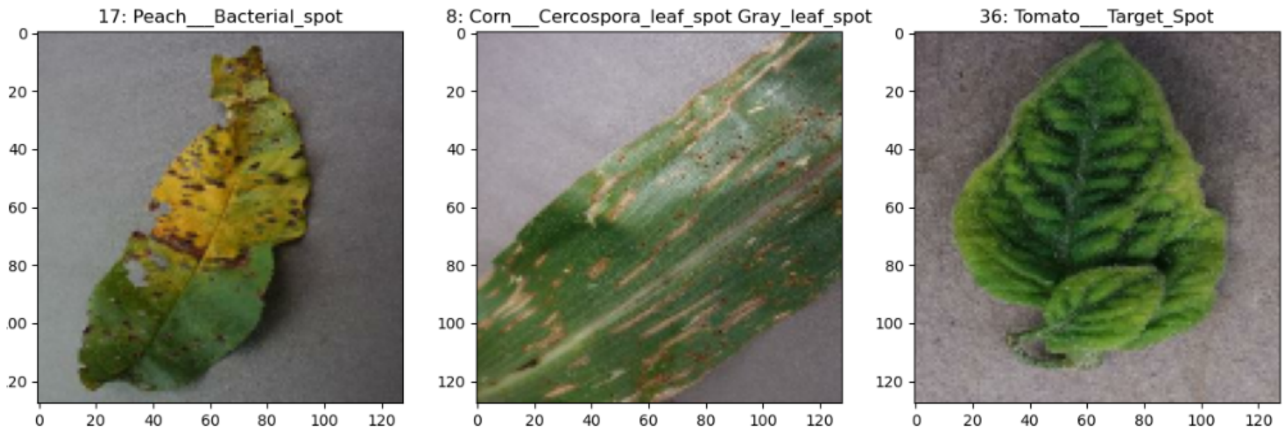

17: Peach___Bacterial_spot
8: Corn___Cercospora_leaf_spot Gray_leaf_spot
36: Tomato___Target_Spot
0
20
40
60
80
100
120

Figure 4. Random images from data augmentation with labels.

Bootstrapping and Bagging

In this work, data ensembling with bootstrapping technique was implemented on the training data. Two types of bootstrapping were tested:

- Bootstrapping (on raw data) down to 200 samples for each class

- Bootstrapping (on data with augmentation) up to 900 samples for each class

Bootstrapping was combined with bagging. Each bootstrapping above was implemented 5 times and 5 models were built using each bootstrapped data. The ensemble the 5 models by a majority vote approach.

Data Pre-processing

Pre-processing on image data was kept simple for this work. Try to decrease the computational cost but maintain the key features in the original data as much as possible. Figure 5 shows the step-by-step pre-processing: resizing, adjusting brightness and contrast and scaling.

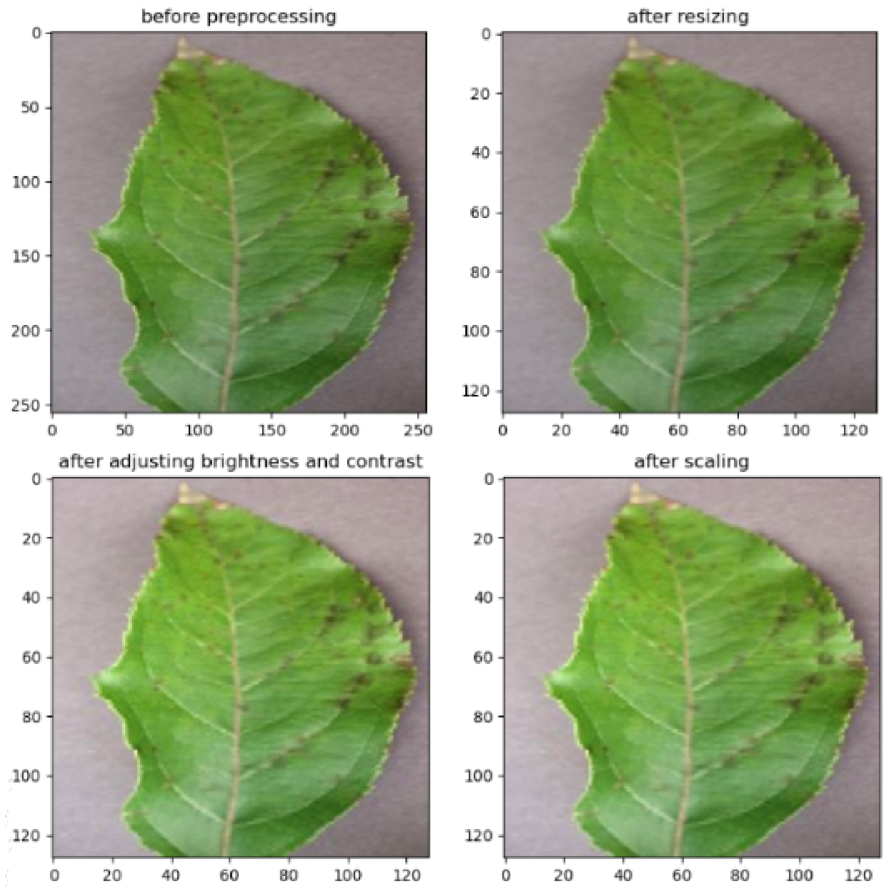
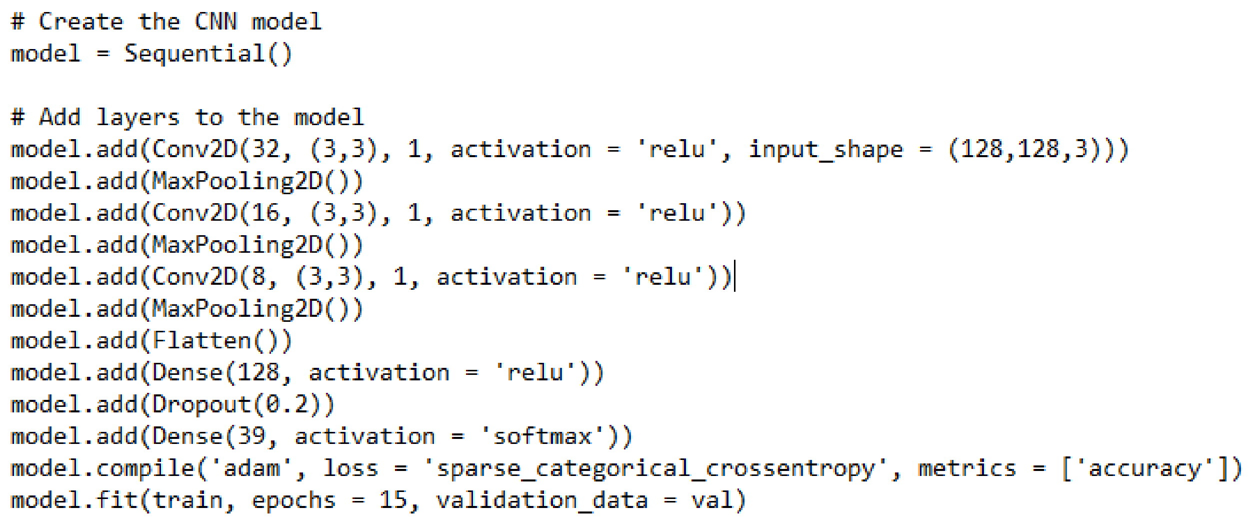


Figure 5. Step-by-step pre-processing: resizing->adjusting brightness-> scaling.

Modeling and Evaluation

CNN models were built using **TensorFlow.Keras.Models**. The original model structure was used here. The focus was to compare model performance of the 4 approaches addressing data issues mentioned in the Data Challenge section.

You can find how the original model was built in the model metric file below:



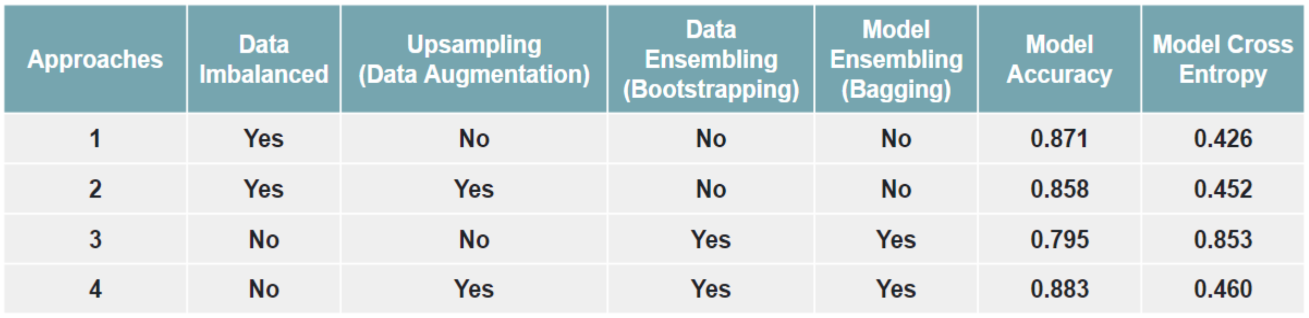
```
# Create the CNN model
model = Sequential()

# Add layers to the model
model.add(Conv2D(32, (3,3), 1, activation = 'relu', input_shape = (128,128,3)))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Conv2D(8, (3,3), 1, activation = 'relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(39, activation = 'softmax'))
model.compile('adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
model.fit(train, epochs = 15, validation_data = val)
```

Accuracy, cross entropy, classification report and confusion matrix were selected as the evaluating metrics. Since classes were imbalanced in testing data, multiple metrics were necessary to accurately determine the superior model performance.

Models built from approach 1 (unbalanced raw data) and approach 4 (augmented data with bootstrapping and bagging to 900) demonstrated the best classification performance. Check Table 2 for accuracy and cross entropy comparison.

Note that for approach 3 and 4, a majority vote was used to generate the final prediction from the 5 different Bootstrapping models. Model 1 showed lower cross entropy while model 4 showed higher accuracy. In order to better evaluate the model performance, classification report and confusion matrix should be investigated to make the call for the winner.



Approaches	Data Imbalanced	Upsampling (Data Augmentation)	Data Ensembling (Bootstrapping)	Model Ensembling (Bagging)	Model Accuracy	Model Cross Entropy
1	Yes	No	No	No	0.871	0.426
2	Yes	Yes	No	No	0.858	0.452
3	No	No	Yes	Yes	0.795	0.853
4	No	Yes	Yes	Yes	0.883	0.460

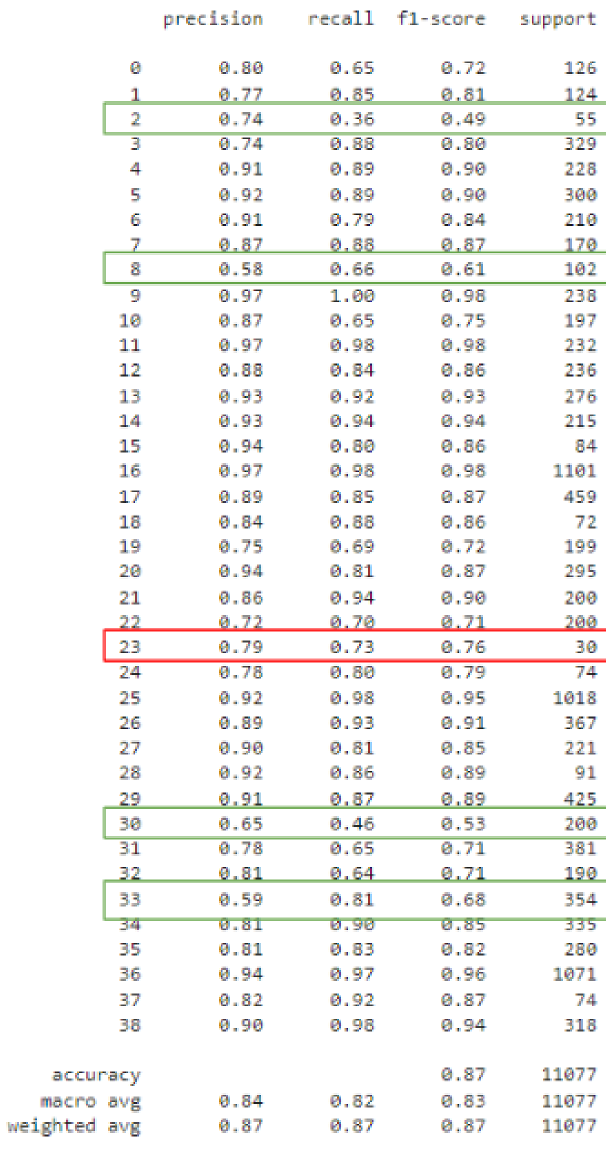
Table 2. Different approaches tested to address data challenges and corresponding model accuracy and cross entropy.

Table 3 is the classification report comparison between model 1 and model 4. Model 1 showed poor predictions (either recall ≤ 0.6 or precision ≤ 0.6) for apple_cedar_apple_rust (2), corn_gray_leaf_spot (8), tomato_early_blight (30) and Tomato_early_blight (33). Model 4 performed better for apple_cedar_apple_rust (2) and Tomato_early_blight (33). For most classes with a limited number of samples, model 4 showed better predictions meaning bootstrapping and bagging can help improve model performance. However, there was too little data, like for Potato_healthy (23), data augmentation degraded the model accuracy.

Model 4 performed slightly better than model 1 by carefully comparing the classification reports. However, the difference is not significant. Model 1 was built using images with the best quality since there was no data augmentation. Model 4 had sufficient and balanced classes but degraded data quality due to augmentation. From the evaluation in this work, it can be concluded that both data quality and quantity play an important role in building an accurate model.

Both model 1 and model 4 performed poorly for two classes, corn_gray_leaf_spot (8) and tomato_early_blight (30). Both models misclassified them, often confusing them with other classes within the same species. The raw data revealed that images from these two classes sometimes depict mild symptoms, making them challenging for the models to detect.

Model 1

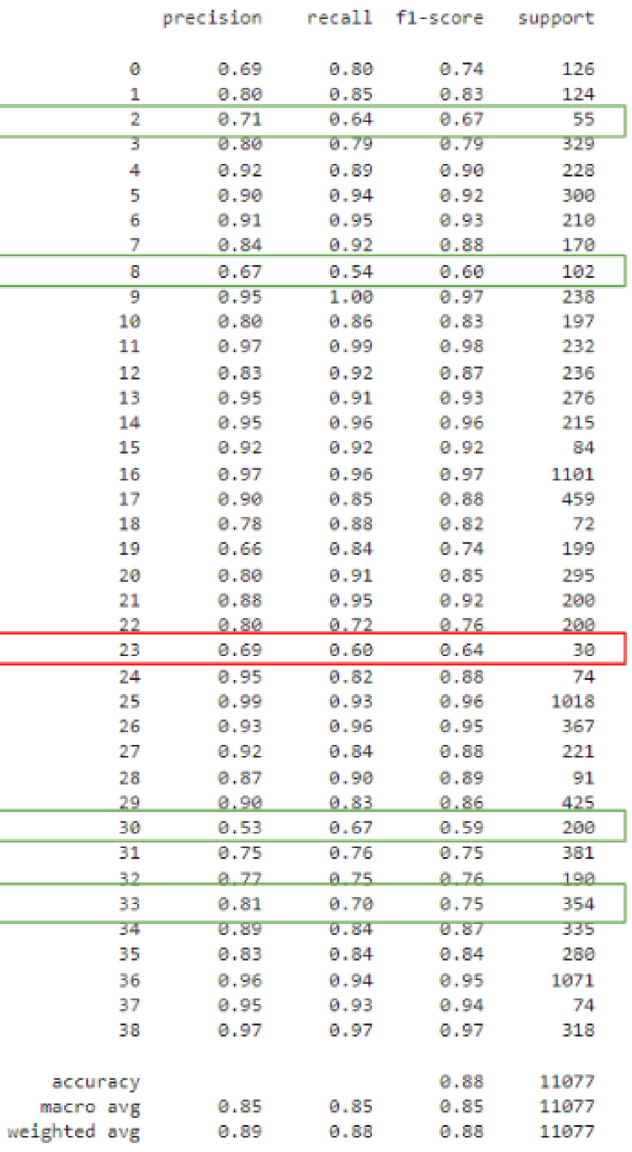


```
              precision    recall  f1-score   support

           0       0.80      0.65      0.72       126
           1       0.77      0.85      0.81       124
           2       0.74      0.36      0.49        55
           3       0.74      0.88      0.80       329
           4       0.91      0.89      0.90       228
           5       0.92      0.89      0.90       300
           6       0.91      0.79      0.84       210
           7       0.87      0.88      0.87       170
           8       0.58      0.66      0.61       102
           9       0.97      1.00      0.98       238
          10       0.87      0.65      0.75       197
          11       0.97      0.98      0.98       232
          12       0.88      0.84      0.86       236
          13       0.93      0.92      0.93       276
          14       0.93      0.94      0.94       215
          15       0.94      0.80      0.86        84
          16       0.97      0.98      0.98      1101
          17       0.89      0.85      0.87       459
          18       0.84      0.88      0.86        72
          19       0.75      0.69      0.72       199
          20       0.94      0.81      0.87       295
          21       0.86      0.94      0.90       200
          22       0.72      0.70      0.71       200
          23       0.79      0.73      0.76        30
          24       0.78      0.80      0.79        74
          25       0.92      0.98      0.95      1018
          26       0.89      0.93      0.91       367
          27       0.90      0.81      0.85       221
          28       0.92      0.86      0.89        91
          29       0.91      0.87      0.89       425
          30       0.65      0.46      0.53       200
          31       0.78      0.65      0.71       381
          32       0.81      0.64      0.71       190
          33       0.59      0.81      0.68       354
          34       0.81      0.90      0.85       335
          35       0.81      0.83      0.82       280
          36       0.94      0.97      0.96      1071
          37       0.82      0.92      0.87        74
          38       0.90      0.98      0.94       318

    accuracy                           0.87     11077
   macro avg       0.84      0.82      0.83     11077
weighted avg       0.87      0.87      0.87     11077
```

Model 4



```
              precision    recall  f1-score   support

           0       0.69      0.80      0.74       126
           1       0.80      0.85      0.83       124
           2       0.71      0.64      0.67        55
           3       0.80      0.79      0.79       329
           4       0.92      0.89      0.90       228
           5       0.90      0.94      0.92       300
           6       0.91      0.95      0.93       210
           7       0.84      0.92      0.88       170
           8       0.67      0.54      0.60       102
           9       0.95      1.00      0.97       238
          10       0.80      0.86      0.83       197
          11       0.97      0.99      0.98       232
          12       0.83      0.92      0.87       236
          13       0.95      0.91      0.93       276
          14       0.95      0.96      0.96       215
          15       0.92      0.92      0.92        84
          16       0.97      0.96      0.97      1101
          17       0.90      0.85      0.88       459
          18       0.78      0.88      0.82        72
          19       0.66      0.84      0.74       199
          20       0.80      0.91      0.85       295
          21       0.88      0.95      0.92       200
          22       0.80      0.72      0.76       200
          23       0.69      0.60      0.64        30
          24       0.95      0.82      0.88        74
          25       0.99      0.93      0.96      1018
          26       0.93      0.96      0.95       367
          27       0.92      0.84      0.88       221
          28       0.87      0.90      0.89        91
          29       0.90      0.83      0.86       425
          30       0.53      0.67      0.59       200
          31       0.75      0.76      0.75       381
          32       0.77      0.75      0.76       190
          33       0.81      0.70      0.75       354
          34       0.89      0.84      0.87       335
          35       0.83      0.84      0.84       280
          36       0.96      0.94      0.95      1071
          37       0.95      0.93      0.94        74
          38       0.97      0.97      0.97       318

    accuracy                           0.88     11077
   macro avg       0.85      0.85      0.85     11077
weighted avg       0.89      0.88      0.88     11077
```

Table 3. Classification report for model 1 and model 4.

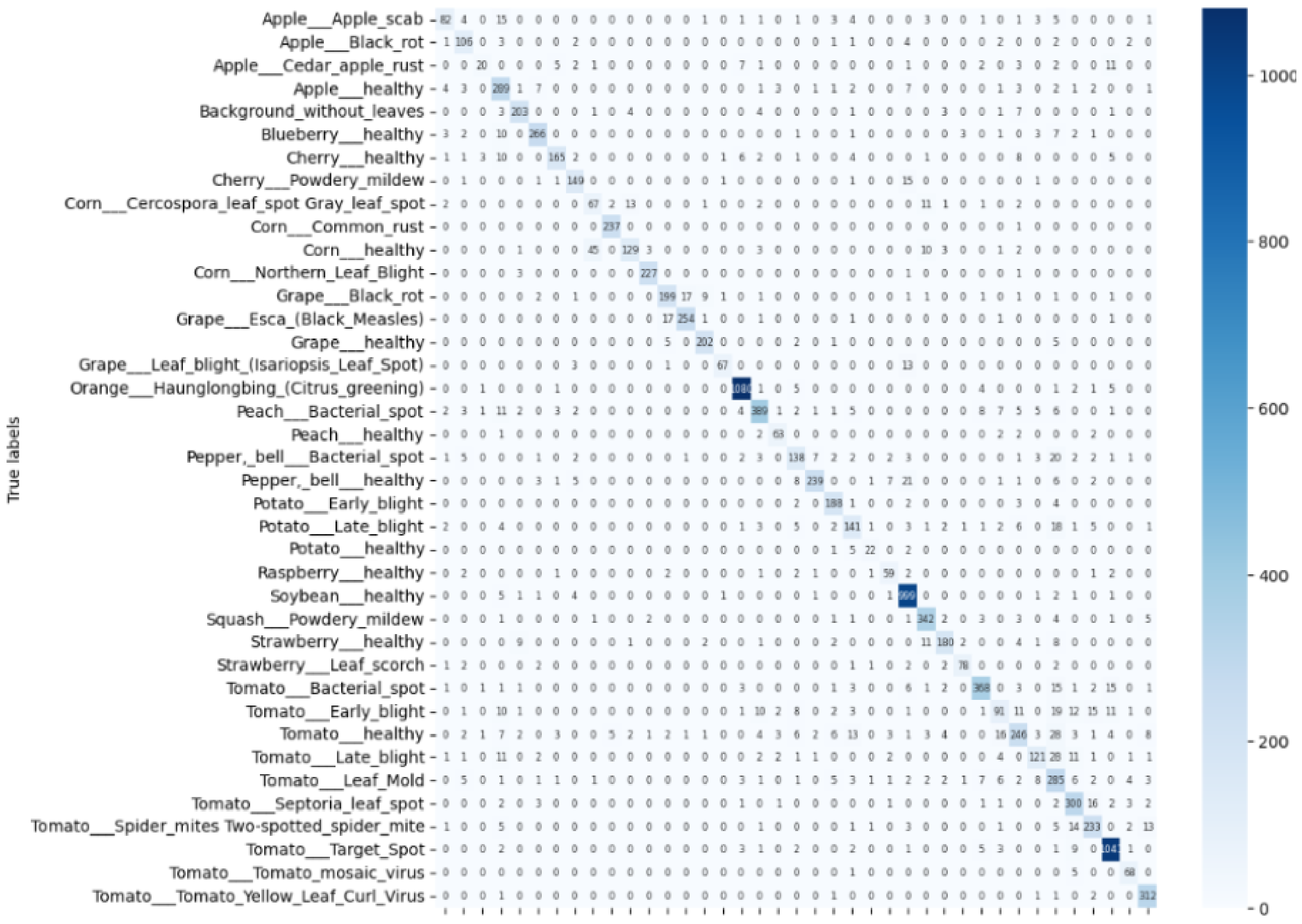


Figure 6. Confusion matrix of model from approach 1.

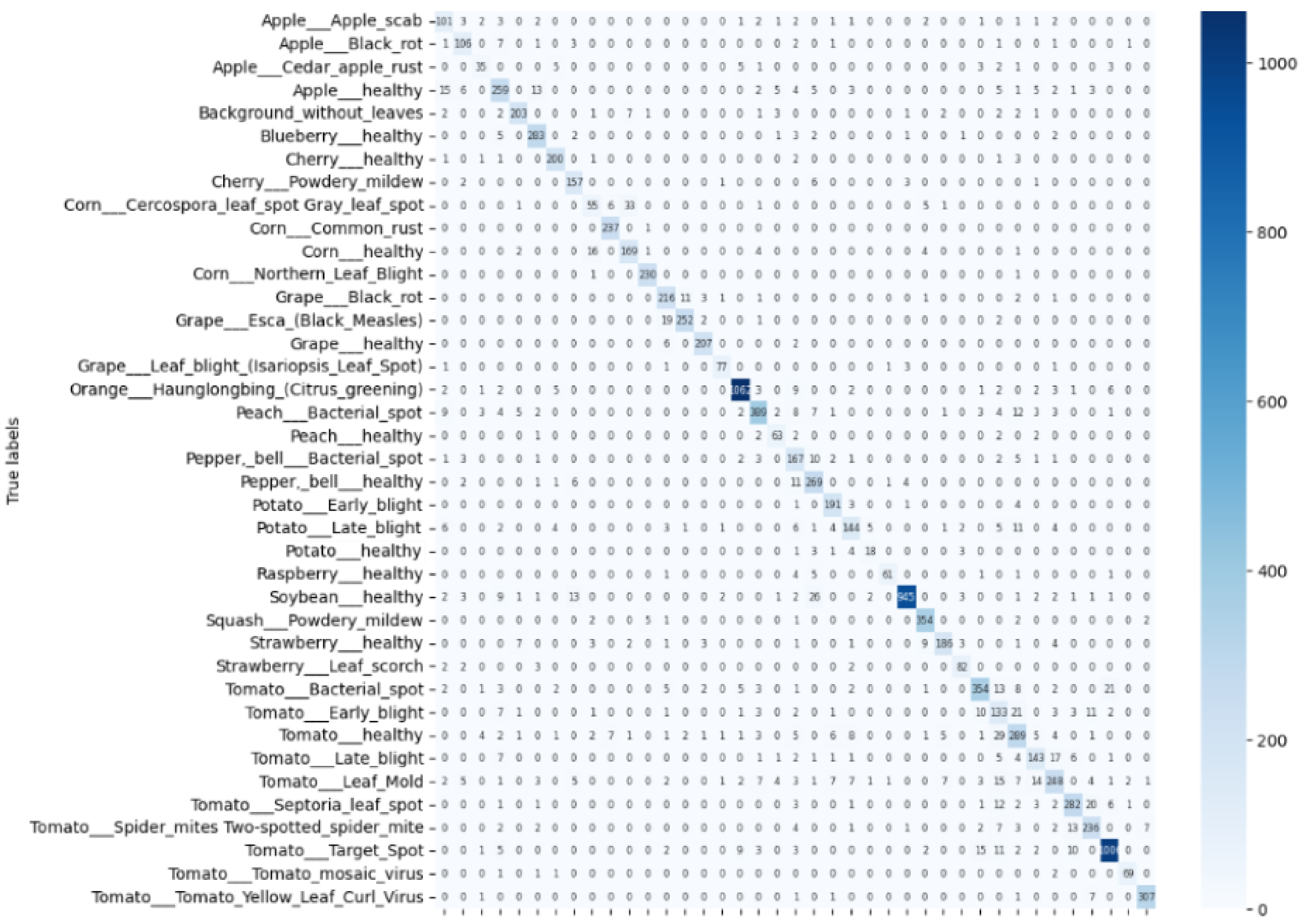


Figure 7. Confusion matrix of model from approach 4.

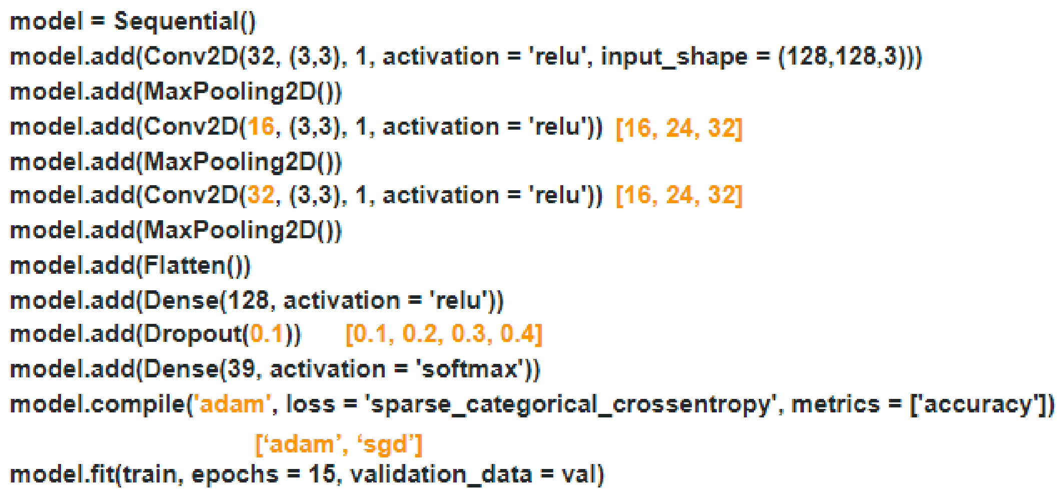
Hyper-parameter Tuning and Evaluation

The hyperband optimization algorithm was implemented in this work to tune the model structure for better predictions. The hyperband optimization algorithm performed the tuning in an efficient way regarding computational resources. It splitted the total possible configurations into buckets based on available resources and then trained a few epochs for each configuration first. After the first round, it only selected top several models for next round training with more epochs until reaching the best one. **Keras-Tuner** package can perform hyperband tuning automatically for CNN models.

Due to limited of resource, only the following parameters were selected for tuning:

- Number of filters in the 2nd convolution layer
- Number of filters in the 3rd convolution layer
- Dropout
- Optimizer for compile

Values tried for each hyper-parameters and the optimized ones are highlighted below:

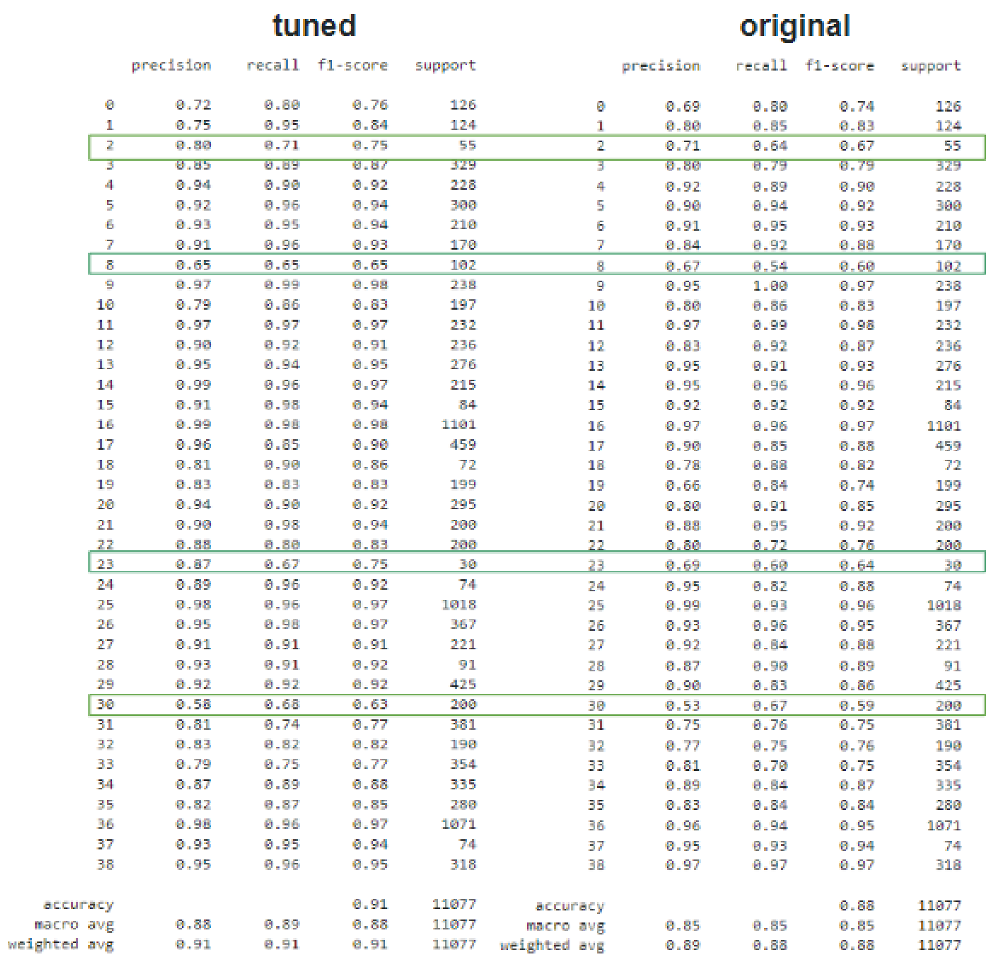


```
model = Sequential()
model.add(Conv2D(32, (3,3), 1, activation = 'relu', input_shape = (128,128,3)))
model.add(MaxPooling2D())
model.add(Conv2D(16, (3,3), 1, activation = 'relu'))  [16, 24, 32]
model.add(MaxPooling2D())
model.add(Conv2D(32, (3,3), 1, activation = 'relu'))  [16, 24, 32]
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.1))     [0.1, 0.2, 0.3, 0.4]
model.add(Dense(39, activation = 'softmax'))
model.compile('adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
                         ['adam', 'sgd']
model.fit(train, epochs = 15, validation_data = val)
```

Training data 4 was used since it performed the best for the original model structure. Original model 4 and tuned model 4 were evaluated by various multi-classification metrics. Accuracy, cross entropy and AUC (one over rest) are provided in Table 4. Classification report comparisons are shown in Table 5 and confusion matrix for the tuned model is shown in Figure 8. Compared to the original CNN model, the tuned model demonstrated better performance with less misclassification for almost all classes in the test data.

Models	Accuracy	Cross Entropy	AUC (one over rest)
original	0.896	0.450	0.996
tuned	0.909	0.441	0.997

Table 4. Accuracy, cross entropy and AUC for tuned model and original model.



	tuned				original			
	precision	recall	f1-score	support	precision	recall	f1-score	support
0	0.72	0.80	0.76	126	0.69	0.80	0.74	126
1	0.75	0.95	0.84	124	0.80	0.85	0.83	124
2	0.80	0.71	0.75	55	0.71	0.64	0.67	55
3	0.85	0.89	0.87	329	0.80	0.79	0.79	329
4	0.94	0.90	0.92	228	0.92	0.89	0.90	228
5	0.92	0.96	0.94	300	0.90	0.94	0.92	300
6	0.93	0.95	0.94	210	0.91	0.95	0.93	210
7	0.91	0.96	0.93	170	0.84	0.92	0.88	170
8	0.65	0.65	0.65	102	0.67	0.54	0.60	102
9	0.97	0.99	0.98	238	0.95	1.00	0.97	238
10	0.79	0.86	0.83	197	0.80	0.86	0.83	197
11	0.97	0.97	0.97	232	0.97	0.99	0.98	232
12	0.90	0.92	0.91	236	0.83	0.92	0.87	236
13	0.95	0.94	0.95	276	0.95	0.91	0.93	276
14	0.99	0.96	0.97	215	0.95	0.96	0.96	215
15	0.91	0.98	0.94	84	0.92	0.92	0.92	84
16	0.99	0.98	0.98	1101	0.97	0.96	0.97	1101
17	0.96	0.85	0.90	459	0.90	0.85	0.88	459
18	0.81	0.90	0.86	72	0.78	0.88	0.82	72
19	0.83	0.83	0.83	199	0.66	0.84	0.74	199
20	0.94	0.90	0.92	295	0.80	0.91	0.85	295
21	0.90	0.98	0.94	200	0.88	0.95	0.92	200
22	0.88	0.80	0.83	200	0.80	0.72	0.76	200
23	0.87	0.67	0.75	30	0.69	0.60	0.64	30
24	0.89	0.96	0.92	74	0.95	0.82	0.88	74
25	0.98	0.96	0.97	1018	0.99	0.93	0.96	1018
26	0.95	0.98	0.97	367	0.93	0.96	0.95	367
27	0.91	0.91	0.91	221	0.92	0.84	0.88	221
28	0.93	0.91	0.92	91	0.87	0.90	0.89	91
29	0.92	0.92	0.92	425	0.90	0.83	0.86	425
30	0.58	0.68	0.63	200	0.53	0.67	0.59	200
31	0.81	0.74	0.77	381	0.75	0.76	0.75	381
32	0.83	0.82	0.82	190	0.77	0.75	0.76	190
33	0.79	0.75	0.77	354	0.81	0.70	0.75	354
34	0.87	0.89	0.88	335	0.89	0.84	0.87	335
35	0.82	0.87	0.85	280	0.83	0.84	0.84	280
36	0.98	0.96	0.97	1071	0.96	0.94	0.95	1071
37	0.93	0.95	0.94	74	0.95	0.93	0.94	74
38	0.95	0.96	0.95	318	0.97	0.97	0.97	318
accuracy			0.91	11077			0.88	11077
macro avg	0.88	0.89	0.88	11077	0.85	0.85	0.85	11077
weighted avg	0.91	0.91	0.91	11077	0.89	0.88	0.88	11077

Table 5. Classification report for tuned model and original model.

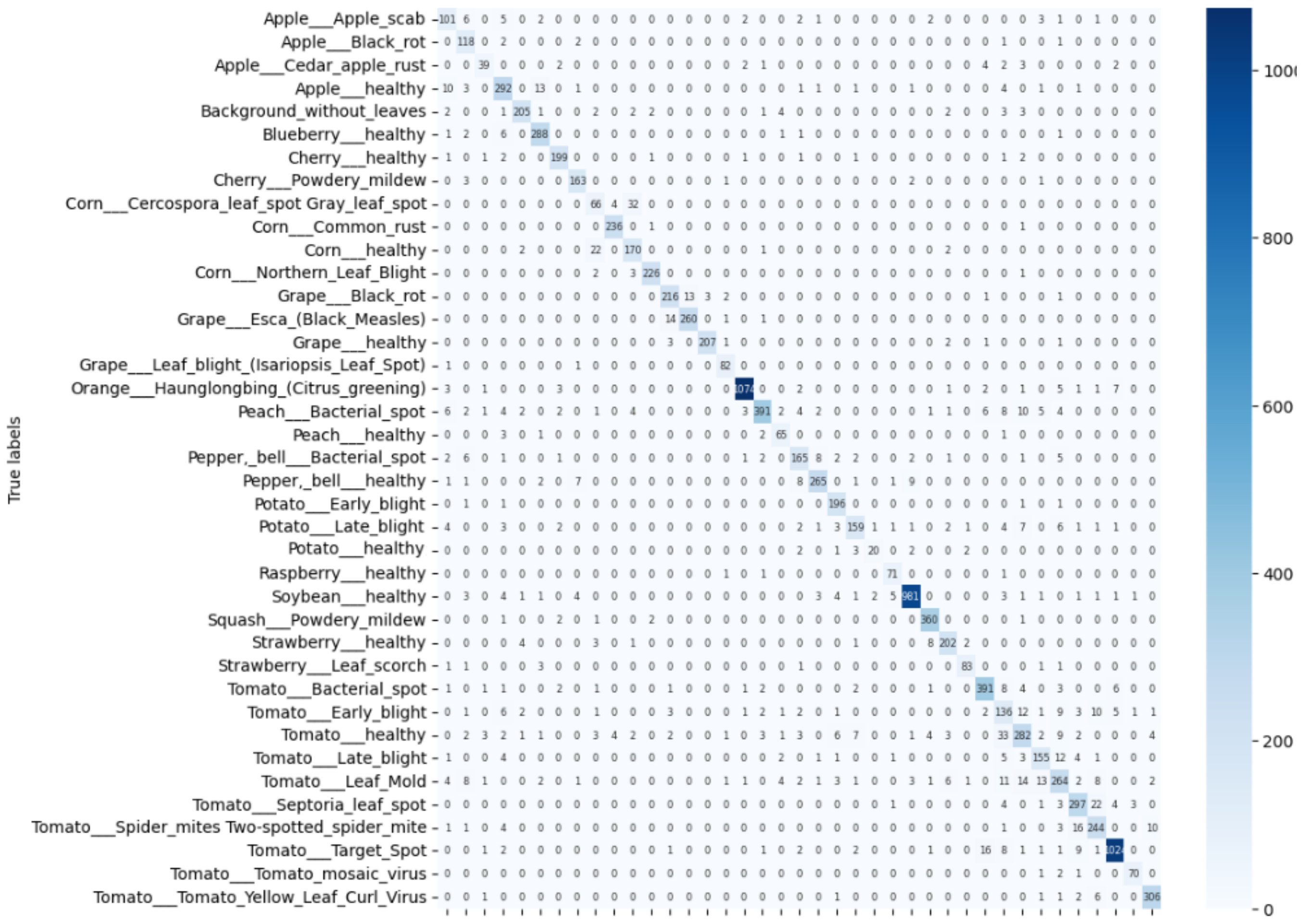


Figure 8. Confusion matrix of the tuned model.

Takeaways

Here are some key points I learnt through this task:

- CNN model works reasonably well for image classification even with simple structure and parameters.
- The hyperband optimization algorithm helps improve CNN model performance by tuning hyper-parameters in a more resource-efficient way.
- Both image quality and quantity are important to building an accurate model.

- Bootstrapping and bagging techniques help increase model performance by mitigating class imbalance issues and reducing prediction errors.

Next Steps

There is so much more to improve for this task. Some are listed below for each section in the whole model building cycle.

Modeling:

More hyper-parameter tuning, such as more filters in the convolution layers, more units in the fully connected layer, learning rate in optimization method, epoch numbers etc.

Feature Engineering:

- Bootstrapping to draw more images from data without augmentation, such as to draw 500 images for each class.
- If improvement is observed for the step above, try drawing more images for each class to find the sweet spot.
- Bootstrapping to draw more images from data with augmentation, say 1500 for each class to see if there are any benefits.
- Try more ensemble times, especially when bootstrapping sample size is small.

Pre-processing:

After settling down on the model and feature engineering part, try using gray scale images as the input to reduce computational cost.

Data Augmentation:

- Multiple data augmentation methods were used in this work, such as rotation, flip, scaling, adding noises, etc. Try to limit to one augmentation method to see if certain augmentation improves model performance.
- Explore other augmentation techniques.

Input Data:

Search online to collect more images for the classes with a limited number of samples.