Plant Leaf Diseases Diagnosis with Deep Learning

Fei Wang
09/21/2023

Why This Project?

Automated Agricultural Systems

https://images.pexels.com/photos/96715/pexels-photo-96715.jpeg?cs=srgb&dl=pexels-alejandro-barr%C3%B3n-96715.jpg&fm=jpg

Mobile App for Houseplant Hobbyists

*https://hips.hearstapps.com/hmg-prod/images/man-watering-houseplants-royalty-free-image-1678736179.jpg?crop=0.668xw:1.00xh;0.136xw,0&resize=640:**

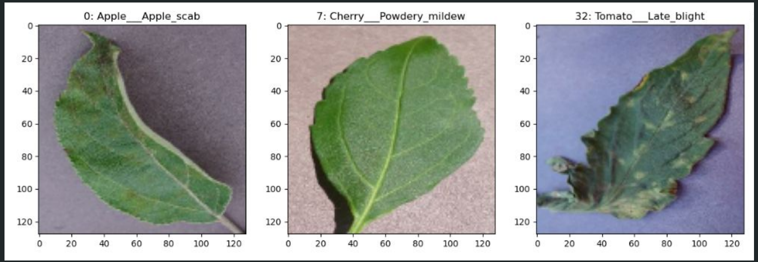
plantix

Agrio

Data

Images of Plant Leaves

- Source: Mendeley Data, originally from a journal paper.[1]
- Total Class: 39, including 1 background (not plants)
- Total Images: 61,486
- Total Species: 15

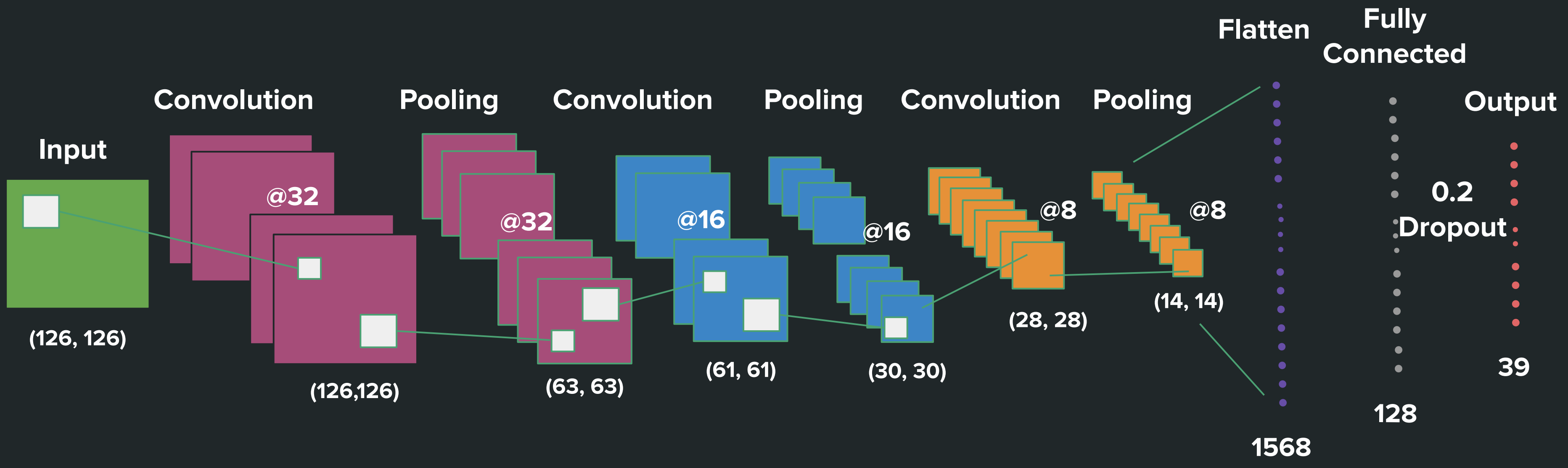


Crops	**Corn**	**Soybean**						
Vegetables	**Pepper**	**Potato**	**Squash**	**Tomato**				
Fruits	**Apple**	**Blueberry**	**Cherry**	**Grape**	**Orange**	**Peach**	**Raspberry**	**Strawberry**

[1] 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

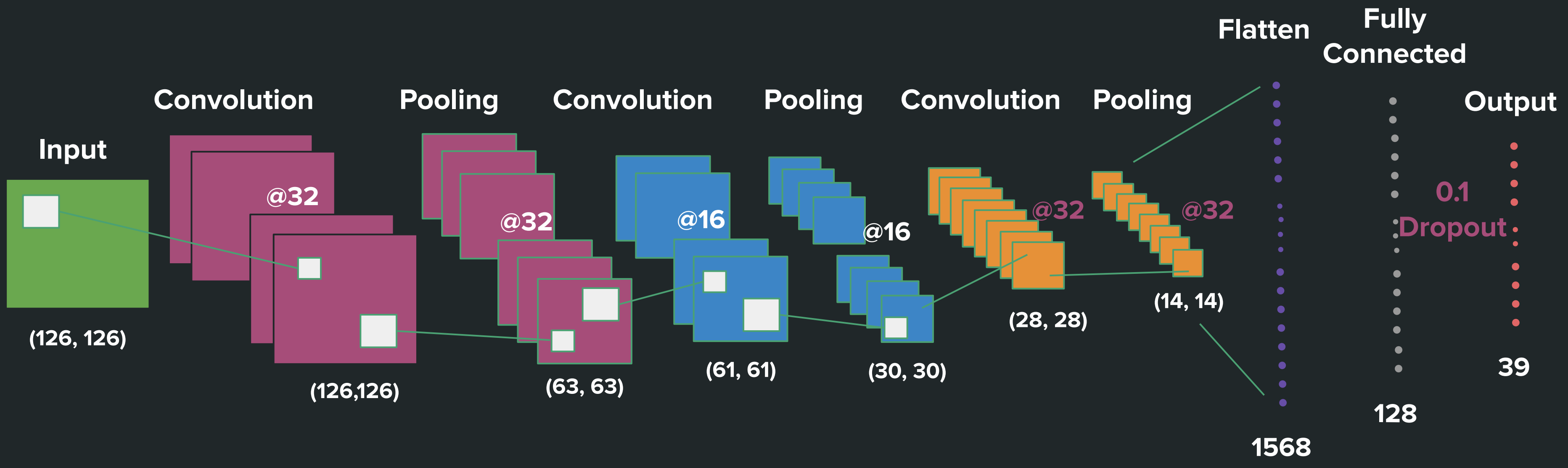
Method

Convolutional Neural Network (original)



A 9-layer structure same as in the journal paper

Convolutional Neural Network (tuned)



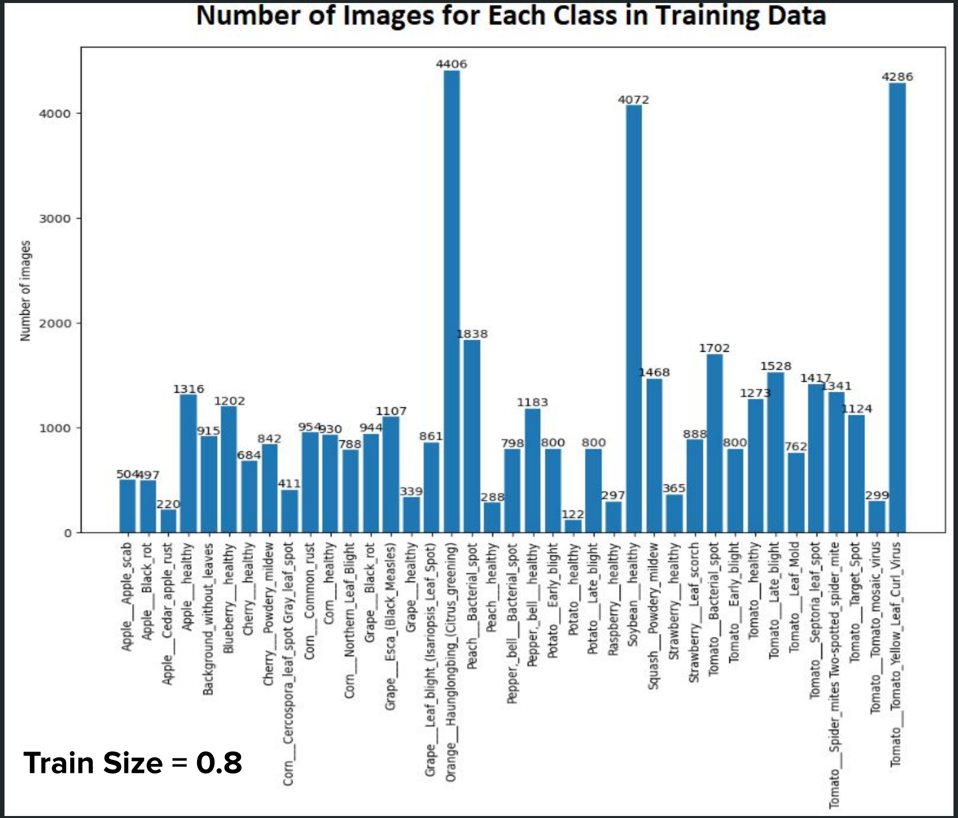
After hyper-parameter tuning

Data Challenges

Imbalance Class

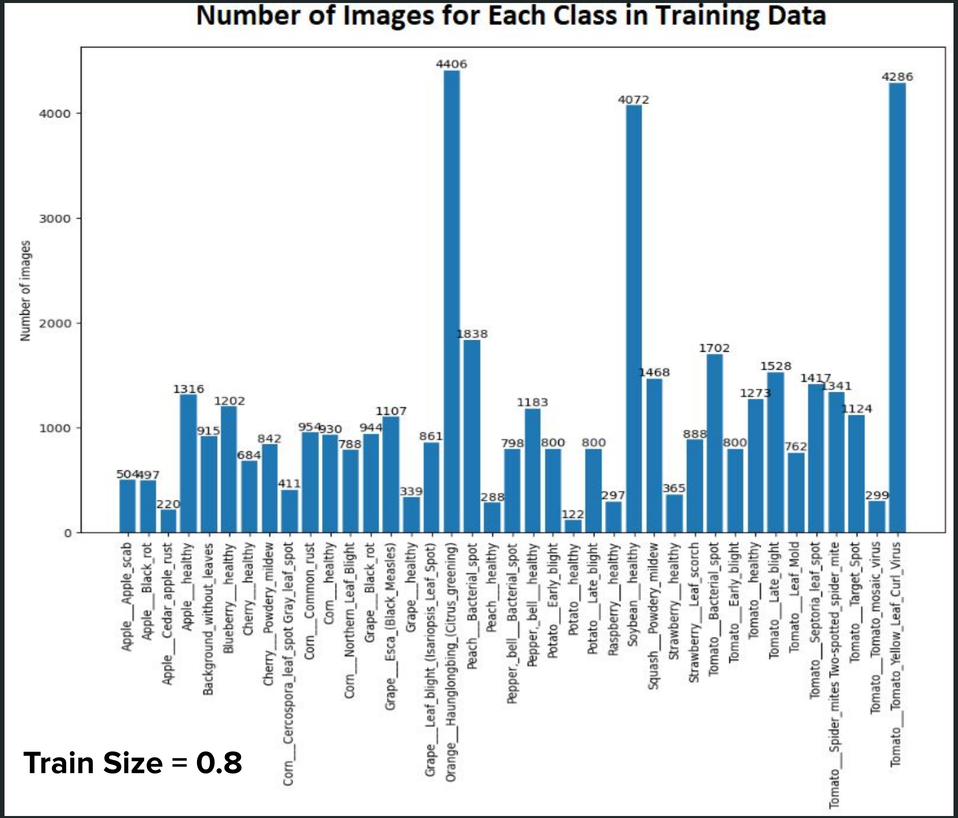
Wavy distribution

Varies from 122 to 4406



Insufficient Data

Potato_heathy only has 122 images in the training data.



Solutions

Hybrid approaches listed below:

Approaches	Upsampling (Data Augmentation*)	Data Ensembling (Bootstrapping)	Model Ensembling (Bagging)	Data Imbalanced	Class Sample Size
1	No	No	No	Yes	122-4406
2	Yes	No	No	Yes	915-4406
3	No	Yes (5 times)	Yes (5 times)	No	200
4	Yes	Yes (5 times)	Yes (5 times)	No	900

* *Data augmentation: rotating, flipping, scaling, adding noises, etc. of existing data.*

Models & Evaluation

Original CNN

Model Metrics (original CNN)

TensorFlow.Keras.Models

```
# create a 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'))
# compile the model
model.compile('adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
# train the model
model.fit(train, epochs = 15, validation_data = val)
```

Model Evaluation

Models	Accuracy	Cross Entropy	Upsampling (Data Augmentation)	Data Ensembling (Bootstrapping)	Model Ensembling (Bagging)	Data Imbalanced	Class Sample Size
1	0.871	0.426	No	No	No	Yes	122-4406
2	0.858	0.452	Yes	No	No	Yes	915-4406
3	0.795*	0.853*	No	Yes (5 times)	Yes (5 times)	No	200
4	0.883*	0.460*	Yes	Yes (5 times)	Yes (5 times)	No	900

** Majority vote was used for Model 3 and Model 4*

Winner: Model 1 & Model 4

Classification Reports

Testing Data also imbalanced

Classification Reports are helpful

Upsampling + Ensembling (Model 4)

Performs better for most classes with less images

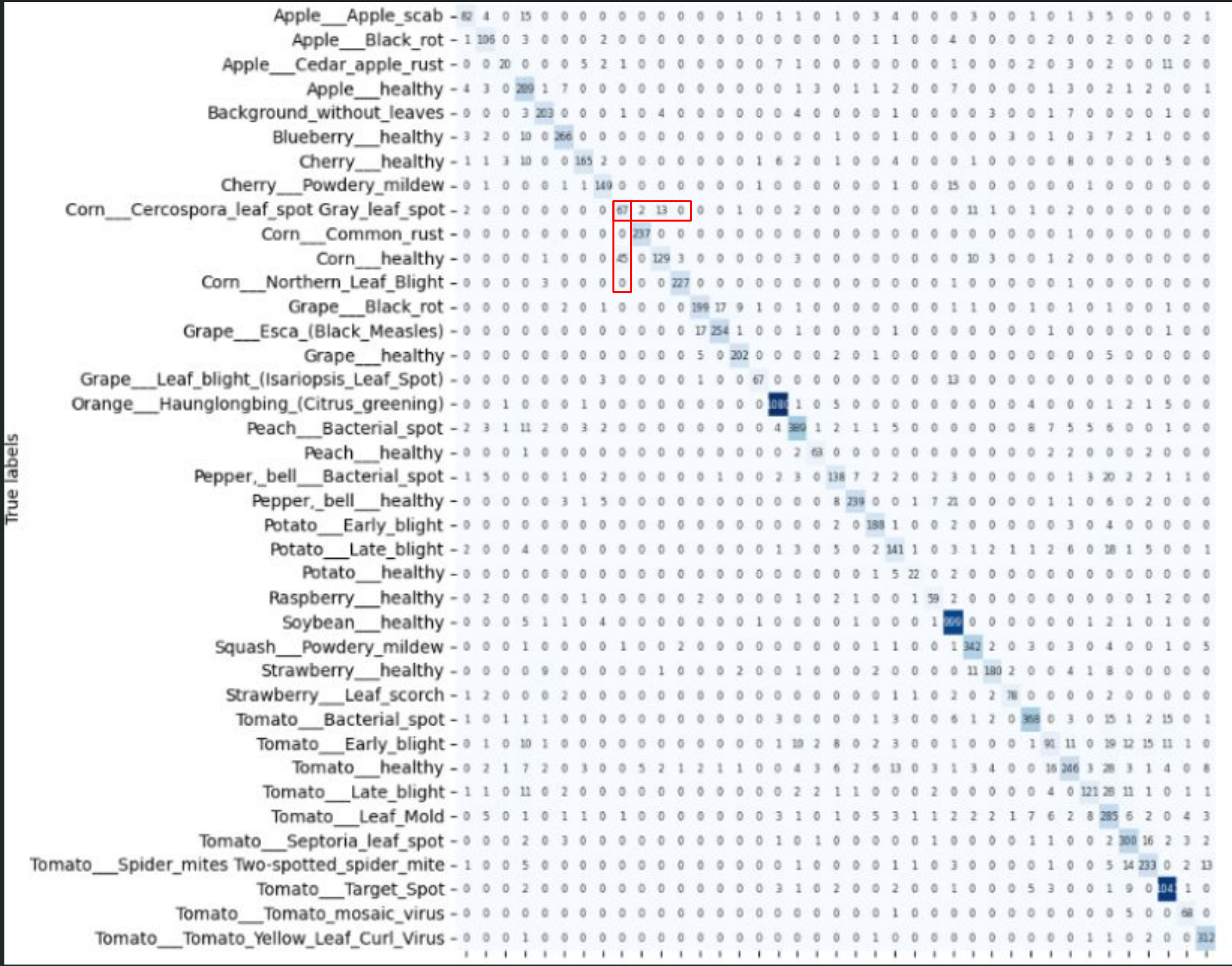
Except for Potato_healthy(23), which has too less images

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

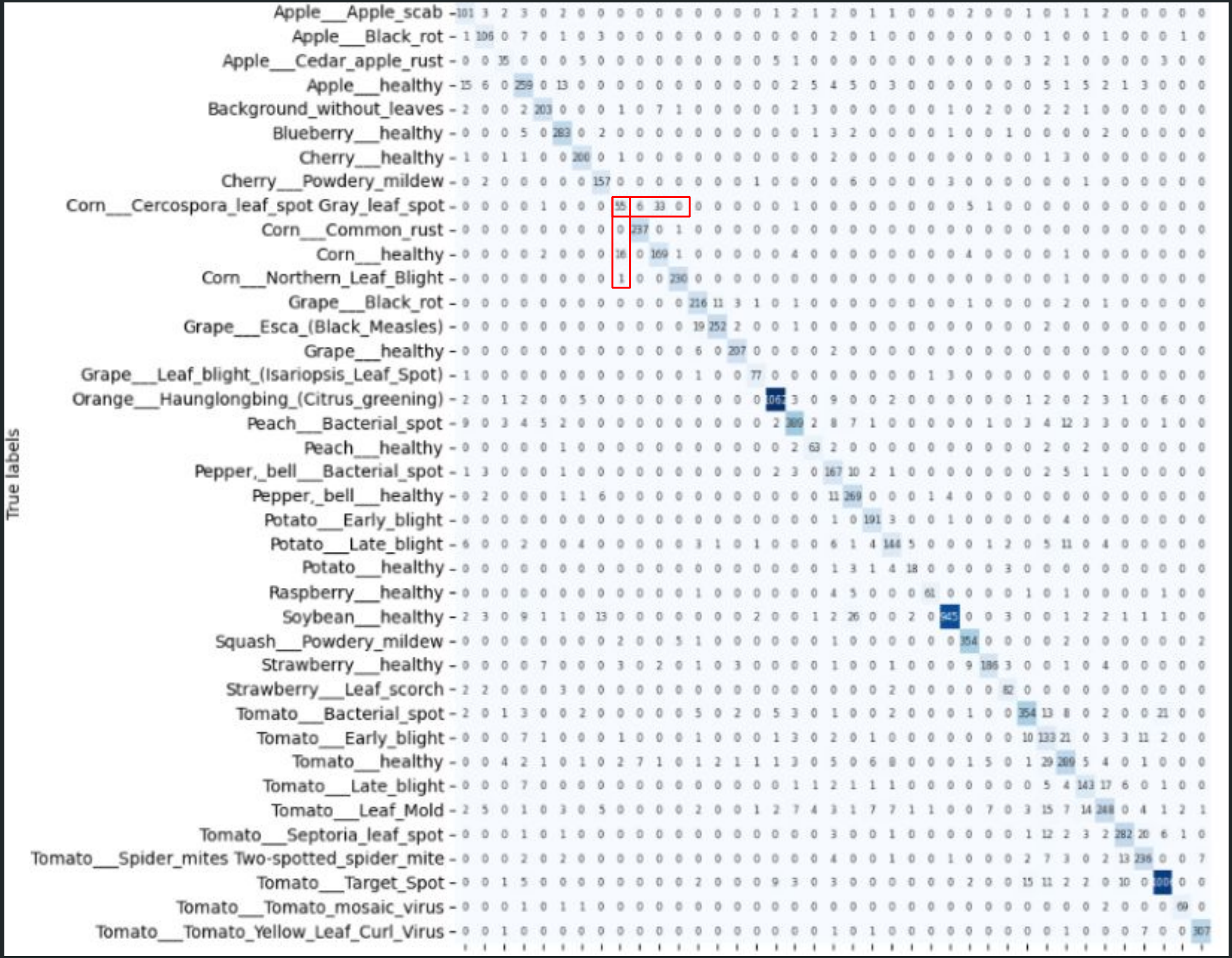
	precision	recall	f1-score	support
0	0.69	0.80	0.74	126
1	0.80	0.85	0.83	124
2	0.71	0.64	0.67	55
3	0.80	0.79	0.79	329
4	0.92	0.89	0.90	228
5	0.90	0.94	0.92	300
6	0.91	0.95	0.93	210
7	0.84	0.92	0.88	170
8	0.67	0.54	0.60	102
9	0.95	1.00	0.97	238
10	0.80	0.86	0.83	197
11	0.97	0.99	0.98	232
12	0.83	0.92	0.87	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



Confusion Matrix

Model 1

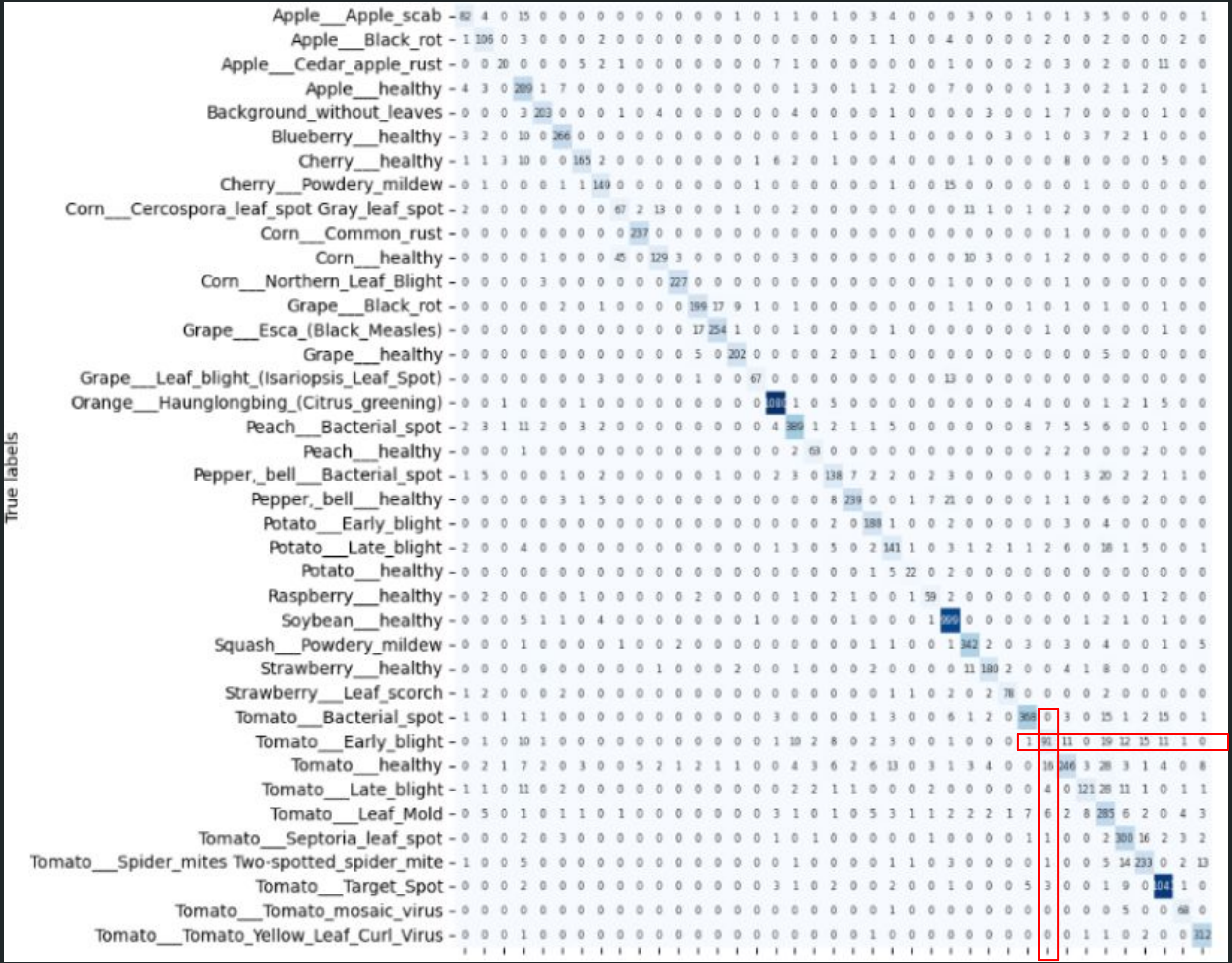
Misclassification within the same species



Confusion Matrix

Model 4

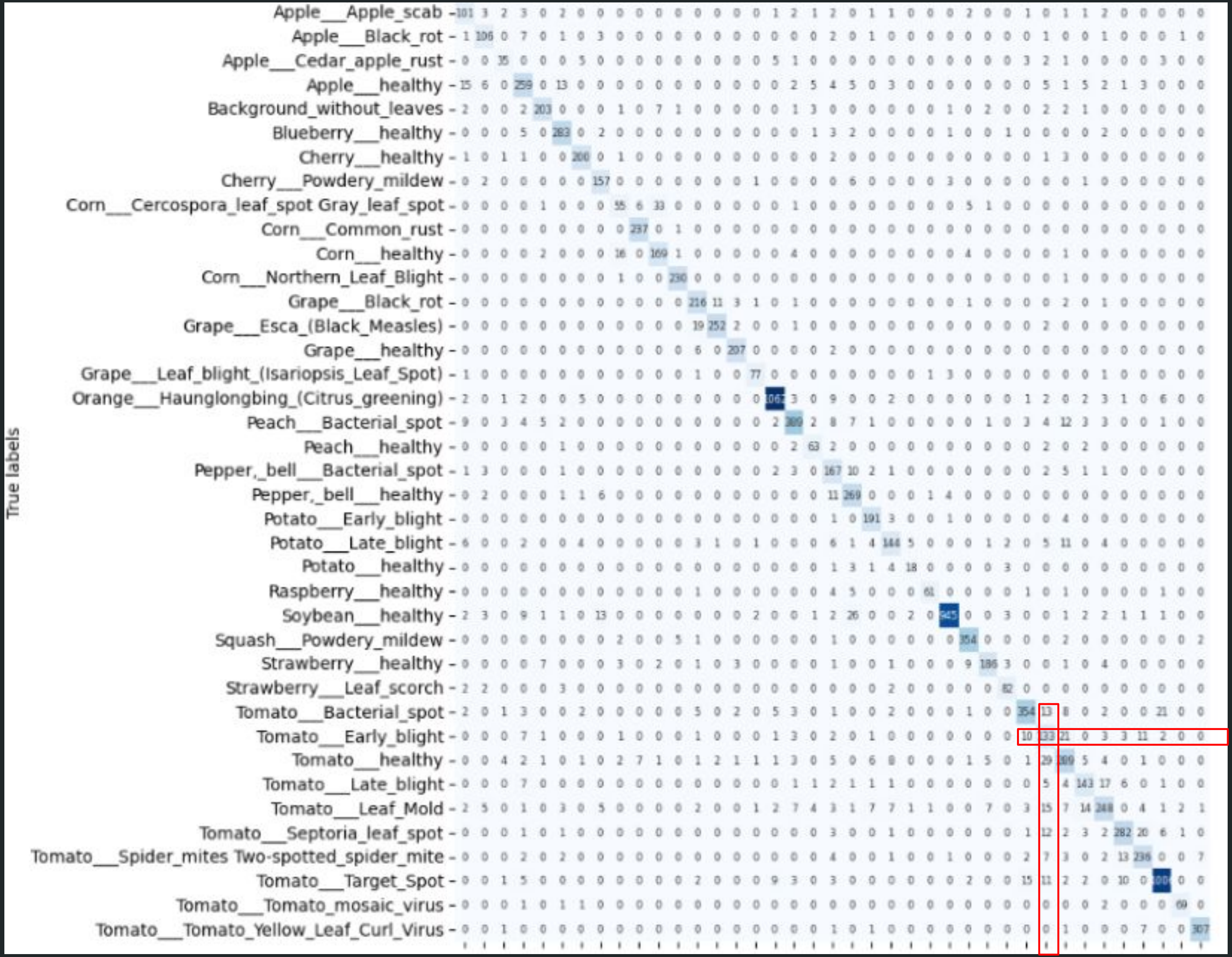
Misclassification within the same species



Confusion Matrix

Model 1

Misclassification within the same species



Confusion Matrix

Model 4

Misclassification within the same species

Models & Evaluation

Tuned CNN on Training Data 4

Model Metrics (tuned CNN)

TensorFlow.Keras.Models

```
# create a 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'))      [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'))
# compile the model
model.compile('adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])   ['adam', 'sgd']
# train the model
model.fit(train, epochs = 15, validation_data = val)
```

keras-tuner
keras_tuner.Hyperband

Model Evaluation

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

** Training data 4 (bootstrapping + augmentation; 900 images for each class) was used for both models.*

Winner: tuned model

Classification Reports

Testing Data also imbalanced

Classification Reports are helpful

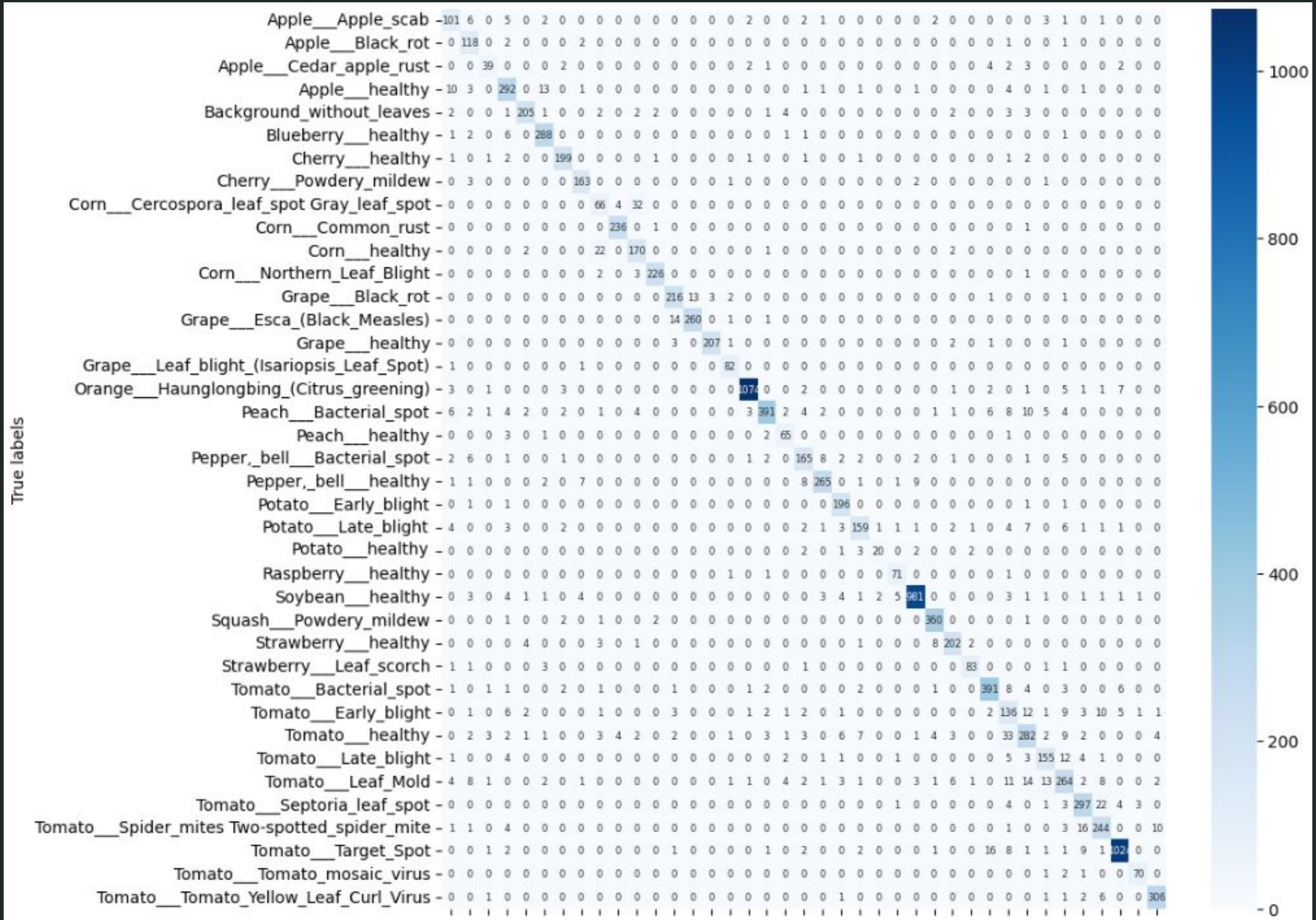
Tuned model performs better than the original one.

tuned

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

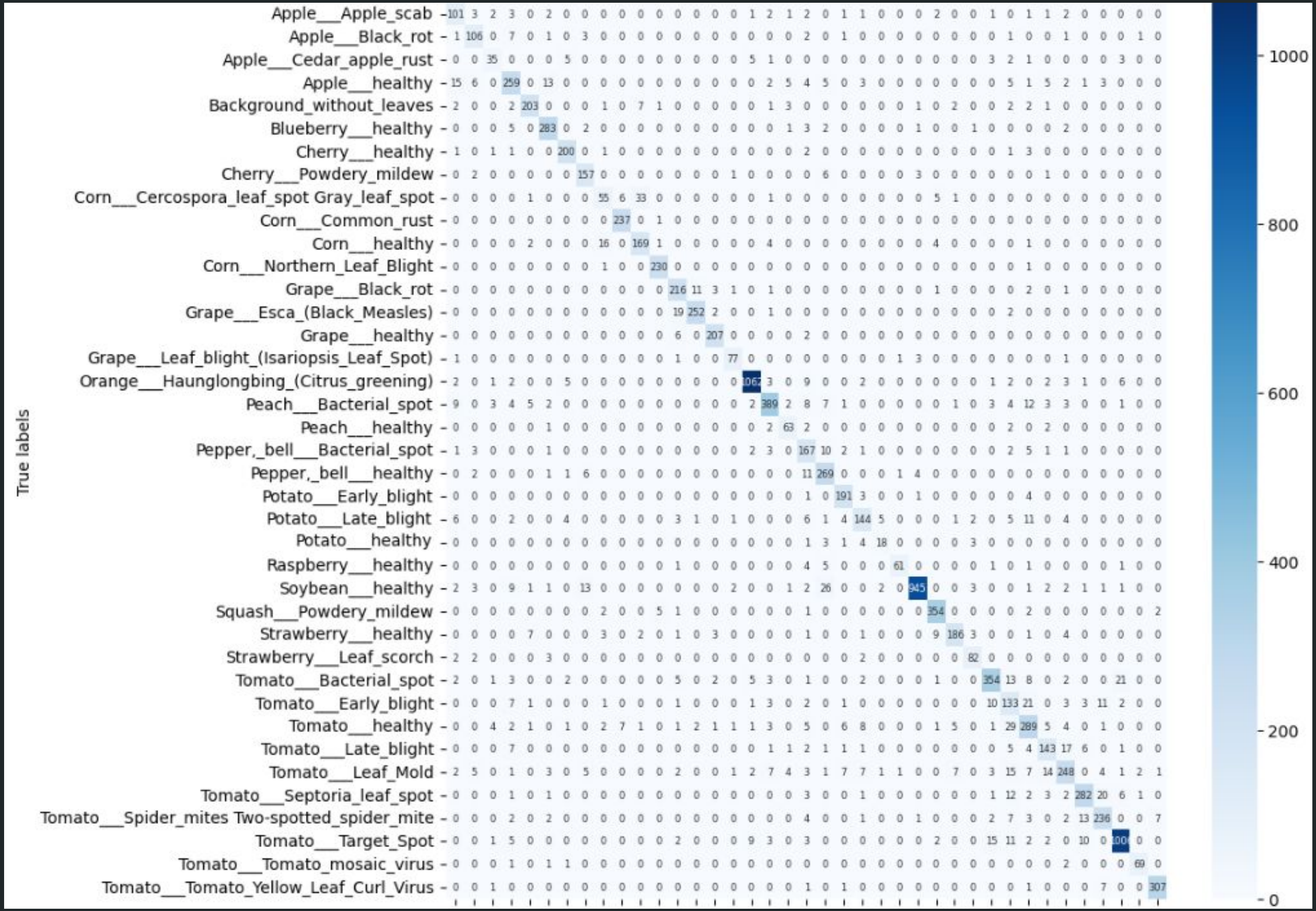
original

	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



Confusion Matrix tuned

Less misclassification



Confusion Matrix original

Takeaways

- Simple CNN model works well
- Hyper-parameter tuning improves model performance
- Image quality and quantity are key
- Bootstrapping and bagging help improve model performance

Next Steps

- Collect more data
- Reduce cost: gray scale?
- Find sweet spot for Bootstrapping sample size
- Is 5-time ensemble a good number
- More hyper-parameter tuning

Acknowledgement

Thanks to Raghunandan Patthar for being a super supporting Springboard mentor and his kindest help.

Thanks to Nicolas Renotte for his fun and engaging YouTube Video on *Build a Deep CNN Image Classifier*.