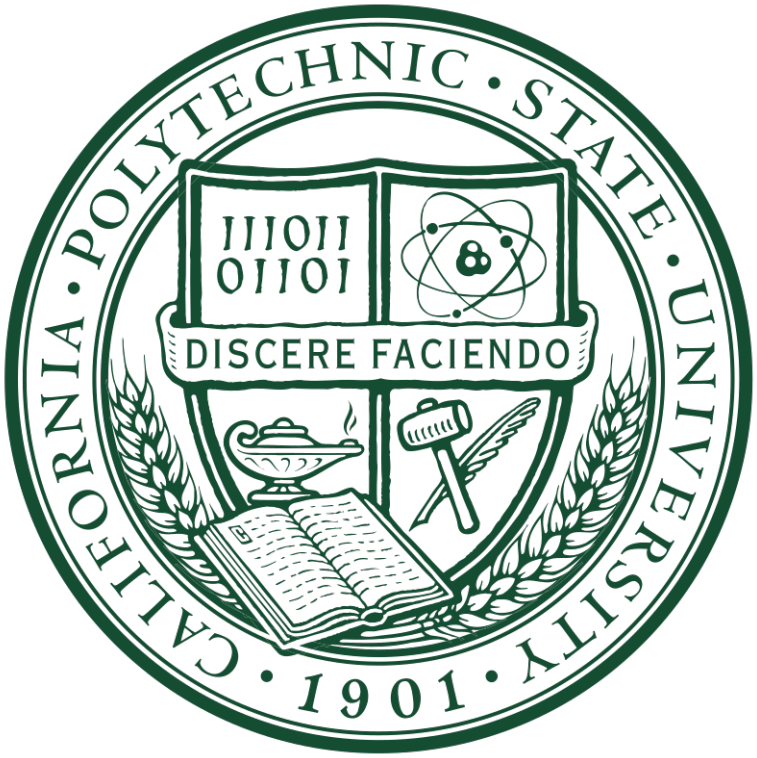
EE428 A comparison of Neural Networks for Leaf Disease Detection

Mathew Shaham

December 11, 2023



Contents

List of Figures

List of Tables

1 Introduction

Plant Disease Detection is important because being able to determine the disease on a plant is not always easy by eye so neural networks and computer vision can help with this task. There are many implementations of plant disease detection, the goals of this project are as follows:

1. Explain methods / papers that have use unique methods for plant disease detection
2. Explain and Utilize one neural network topology
3. View Convolutional Filter Outputs
4. Transfer learning to at least 1 different dataset

The main network utilized will be a Convolutional Neural Network that will be trained on the New Plants Disesases Dataset which is a variation of PlantVillage. Through this project we will discuss different dataset options, how to obtain the datasets in google colab, performing transfer learning, and performance of neural networks (transfer learning and other similar papers). Figure 1 shows the outcome, a correct prediction for each disease class on any input image from the dataset (even ones the classifier has not yet seen, given it falls into one of the categories trained upon). After using this, we will manipulate this network to be used as a framework for a different dataset and determine the performance.
Code for the base project: `https://www.kaggle.com/code/kunjushaji/identification-of-plant-leaf-diseases-using-cnn`

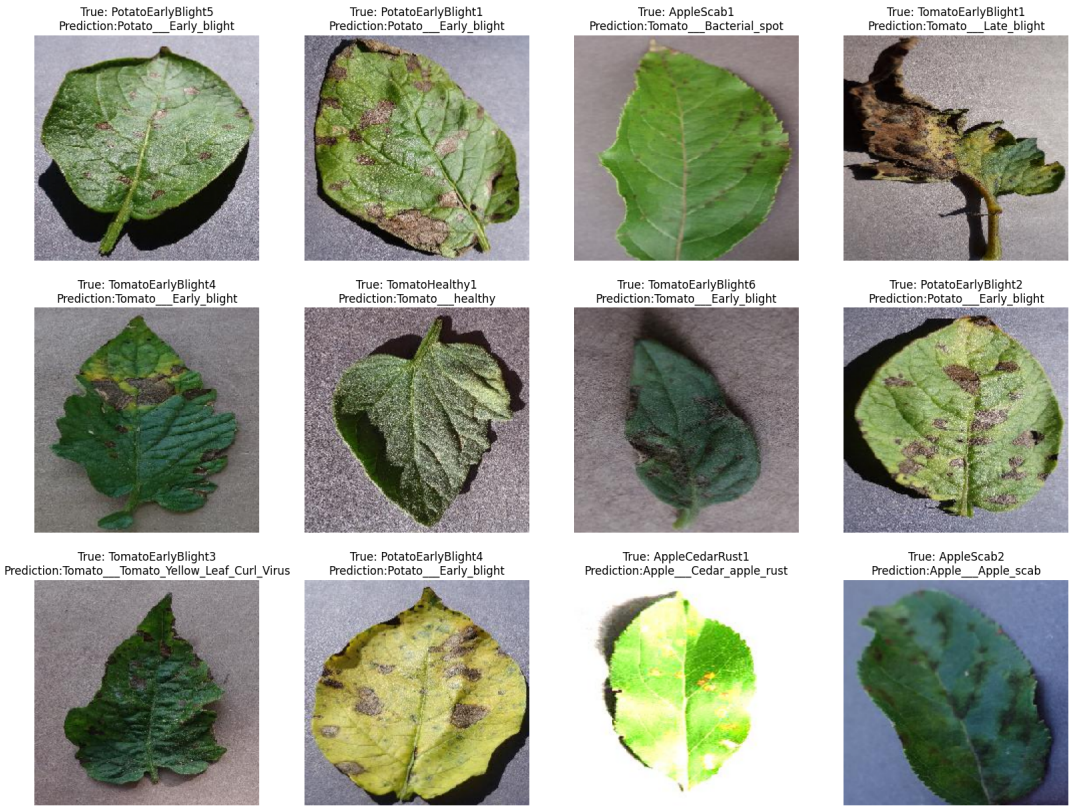


Figure 1: Output after predictions have occurred

2 Previous Work, & Datasets

2.1 Multiple vs. Single Species

Some papers have been implemented that deal with multiple diseases in a single plant. The goal of this paper and many others are to classify a wide range of plants with numerous diseases. However, it is important that many single plant focused papers implement innovative networks with impressive performance metrics. Papers and networks have been introduced for diseases on tomato plants, cucumbers, squash (powdery mildew), and other single species datasets with multiple diseases.

2.2 Previous Work on other Methods of Plant Disease Detection

Popular Transfer Learning models include VGG-16, ResNet, DenseNet, and Inception [4]. Furthermore, J. Andrew et. all in [4] discuss how vanishing gradient issues caused by skip connections were eliminated using batch normalization regularization, but that deeper models have other challenges like over-fitting, covariant shifts, and training time complexity. In their work they compared Inception V4, VGG16, ResNet50, and DenseNet121 using the PlantVillage Dataset showing improved performance and more classes than previous papers using the same models and similar ones (like AlexNet).

In Figure 2 the number of parameters are reduced to around 6 million and the model is applied to 2 large datasets and 3 single plant-multi disease dataset. The model uses a pre-trained VGG16 model with 2 layers of 64 filers and 2 layers of 128 filters. Then outputs this to 3 Inception v7 Convolution blocks with a different number of filters in parallel that get concatenated to 1024 total filers as shown in the figure from Thakur, P. et. al in [1].

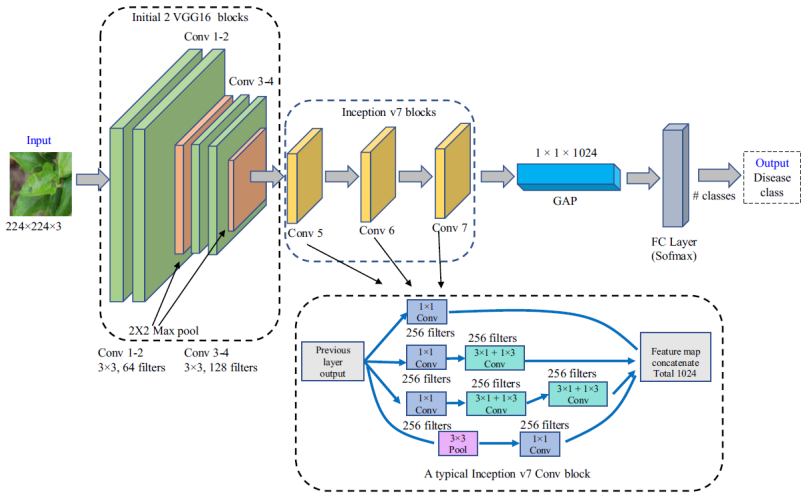


Figure 2: Showing the Pre-Trained VGG16 blocks and the Inceptionv7 blocks with differing number of convolution blocks in parallel that become concatenated. Image Source [1]

Figure 3 shows classes contained in New Plant Disease Dataset which is essentially from plant village dataset used to train the network for this project and the classes and frequency for leaves in FieldPlant Paper [2]. Note that FieldPlant has about 30 classes over 13 plant species compared to 38 classes over greater than 40,000 images in New Plant Disease Dataset.

In Mithu et. al on pages 347 - 357 in [5] Classical Machine learning methods are used including support vector machines (SVM), K-nearest neighbor (KNN), desicion trees, and Naive Bayes on 3 different pumpkin/squash family diseases. They compared these methods to a Convolutional Neural Network (CNN) as well and had from best to worst performance of: CNN (87.67%),

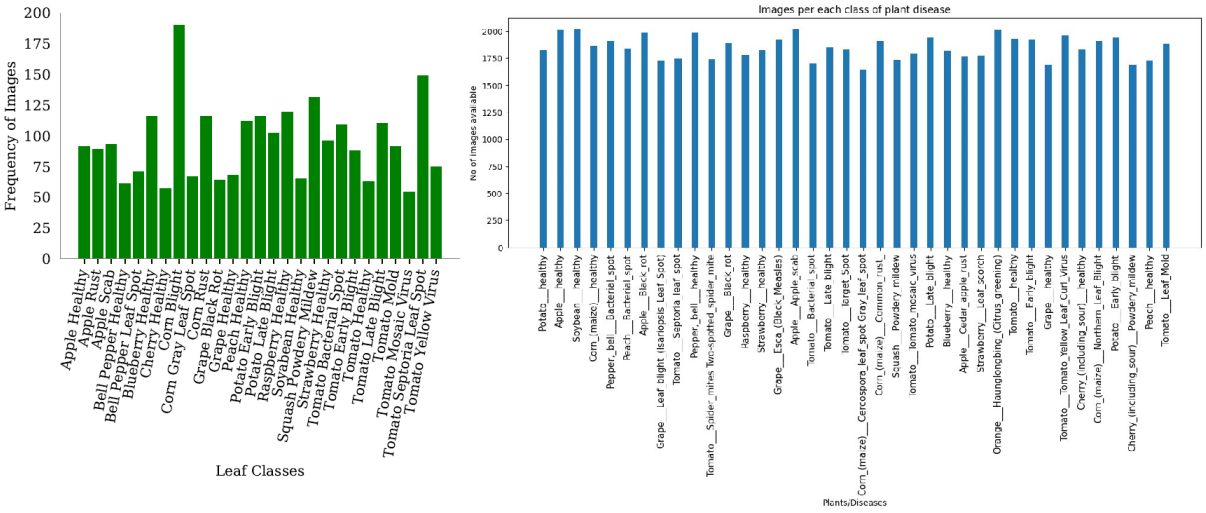


Figure 3: Classes Contained in New Plant Dataset used for model training vs. Field Plant dataset in [2]

KNN, SVM, Naive Bayes, and Decision Tree(69.25%). In Khadizatul A. et al. on pages 359-375 in [5] transfer learning is used to observe and detect cauliflower diseases. What makes this paper unique is the use of a segmented image instead of just resizing and contrast-enhancing. The segmented image is just the disease / features and a map of black or other artifacts making features easier to detect without increased background clutter.

In SSM-Net for Plants Disease Identification in Low Data Regime by Jadon S. [6] Few-shot learning techniques are discussed that of metrics based, models based, and optimization based methods are discussed with a focus on metrics. They use Siamese Networks with 2 parallel layers that learns how to discriminate between 2 inputs. The Stacked-Siamese-Matching (SSM) Net proposed extracts dscriminative features allowing finding differences among diseases. In their paper [6] they found SSM-Net performed better than Trasnfer Learning Methods like VGG for their dataset along with outperforming Matching Networks like miniImageNet.

In LeafGAN: An Effective Data Augmentation Method for Practical Plant Disease Diagnosis by Cap Q. et. al [7] the main points are that datasets can be biased, imbalanced, and networks can tend to overfit since image features are typically smaller than objects in general recognition tasks. This is visible through visualization tools like Gradient Class Activation Map (GradCAM) which will be discussed in the next section. LeafGAN determines the area of the image that is relevant for diagnosis and segments the region of interest (ROI). Label-free leaf segmentation (LFLSeg) is used to define a "partial leaf" class for training which is done through heatmaps defined by something like GradCAM [7]. The main purpose is data augmentation and transforming a diverse range of backgrounds, but is defined only towards cucmbers in the paper.

The final paper reviewed for this project was a review of Common Pre-Trained Models for Image-Based Plant Disease Diagnosis by Dong, X. et al. in [3]. They use DeeplabV3 pre-trained framework utilizing semantic segmentation. Semantic segmentation is a deep learning algorithm that associates a label or category with every pixel in an image. Figure 4 shows the 3 main network topologies used in the paper as pre-trained networks and the resulting outputs.Figure 5 shows the Semantic Segmentation classes that each pixel was placed into, creating a mask of where the disease is. Based on this my guess is the network can determine the corresponding disease class the leaf lies within.

Other papers I would have liked to investigate if time permitted are: Explainable Vision Transformers for CNN Plant Disease Identification by Thakus, P. et. al in [8], Crop leaf disease recognition based on Self-Attention convolutional neural networks (SA-CNN) by Zeng Wihue et. al in [9], and others in the bibliography file of this paper but not on here!

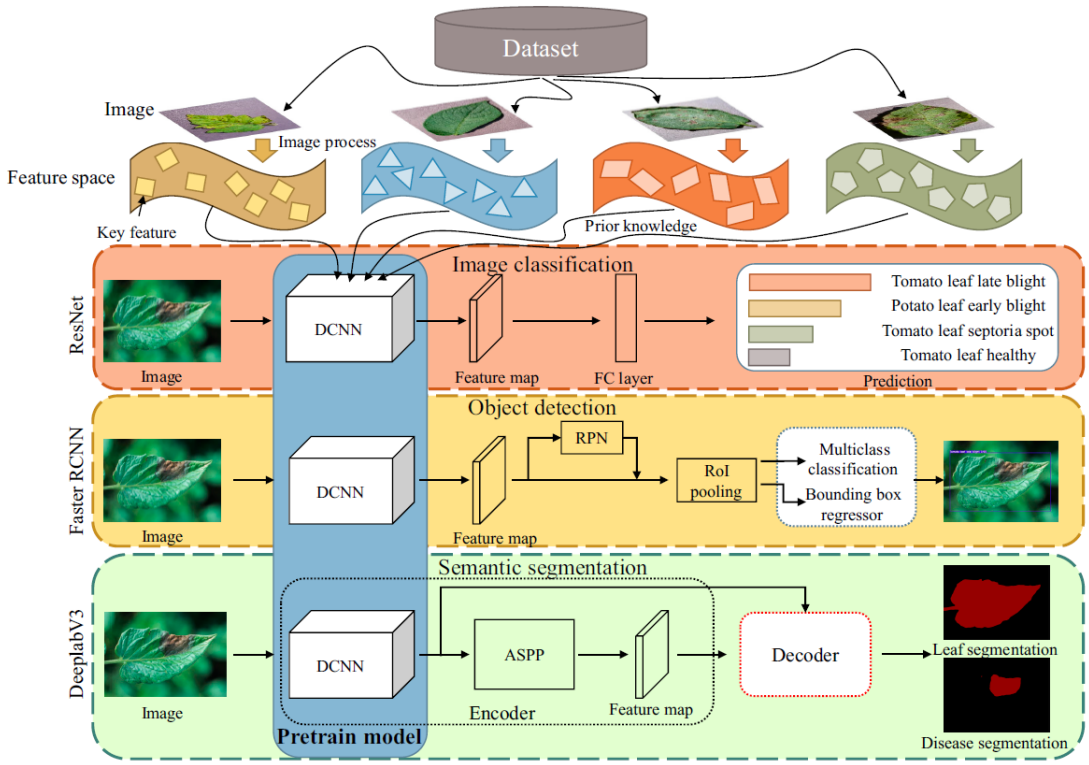


Figure 4: Showing Pre-trained models that can be used to predict plant disease classes [3]

Figure 5: Showing Semantic Segmenetation for different classes of diseased leaves in different shapes with robust performance [3]

2.3 Datasets Available

There are somewhere between 5 - 10 datasets I could find for multiple disease and plant species classifications. I found many more datasets out there in circulation that are aimed toward multiple diseases for single plants.

Table 1: Dataset Summary

Dataset Name	**Number of Images and classes**	**Benefits**	**Disadvantages**
PlantVillage (PV)[1]	Images - 54,303 Classes - 38 (species and disease)	Original / Widely Used	Only Labe Environment Background (not realistic) 3
PlantDoc[2]	Images - 2,569 Classes - 30 (13 plant species)	More Realistic Images	Some Actually Labeled Incorrectly (read in 1 or 2 papers)
FieldPlant[3]	Images - 5,170 Classes - 27 (8,629 individually annotated leaves)	More Realistic Images Improvement on PlantDoc	Unknown
New Plant Disease [4]	Images - around 87,000 Classes - 38 (same as PV) *80/20 training validation split with 33 test images)	Augmented Images adds to scale invariant feature transform similarities (network learns to adapt to rotation and brightness shifts)	Essentially Plant Village with some Image Augmentation
AI Challenger 2018 [5]	Images - 32,739 Classes - 61	Large number of classes with sizable number of images	Not much information on the competition - had to go digging to find correct labels for classes & some classes have few images (less than 10)
Embrapa [6]	Images - 2,326 Classes - 171 (21 plants)	Large number of classes and subdivided images to create 46,513 total (some just zoomed in diseases)	Some images are just the disease meaning still small amount of data (even if augmented)

[1] `https://www.kaggle.com/datasets/emmarex/plantdisease`
[2] `https://public.roboflow.com/object-detection/plantdoc`
[3] `https://universe.roboflow.com/plant-disease-detection/fieldplant/dataset/11`
[4] `https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset`
[5] `https://www.kaggle.com/datasets/jinbao/ai-challenger-pdr2018/data`

[6] https://www.digipathos-rep.cnptia.embrapa.br/jspui/zipsincollection/123456789/3

An example of the differences is clear from Figure 6

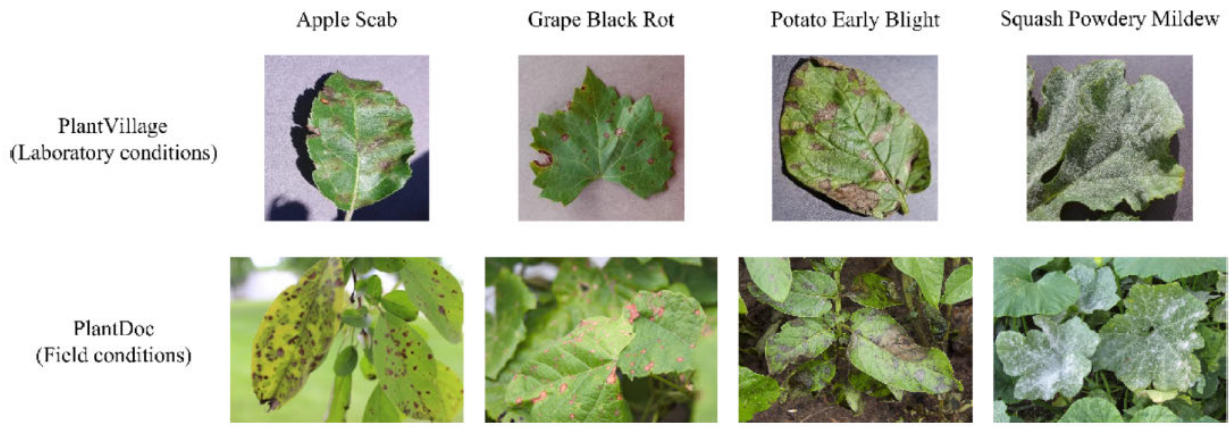


Figure 6: FieldPlant Dataset is in a more realistic setting than the lab-generated photos of New Plant Diseases Dataset

3 My Contributions

3.1 Libraries Used

- Basic Libraries: Numpy, Os, Matplotlib, time, pandas
- Various tensorflow/keras libraries for making the laters and tensorflow tensors along with other necessary supporting functionality.
- Other Libraries: shown below
- !pip install h5py & from tensorflow.keras.models import load_model
- import keras.callbacks as callbacks # for setting model parameters (optimizer)
- !pip install tf-keras-vis
- tf-keras-vis for GradCAM, Maximization Activation, and other visualization tools

```
!pip install -q opendatasets
!pip install plotly

# to download datasets from kaggle for AI Challenger and New Plant Diseases
import opendatasets as od
from google.colab import files
import plotly.graph_objects as go_obj

# imports from tf-keras-vis
from tf_keras_vis.activation_maximization import ActivationMaximization
from tf_keras_vis.activation_maximization.callbacks import Progress
from tf_keras_vis.utils.scores import CategoricalScore
# extract intermediate layer and have output at linear
from tf_keras_vis.utils.model_modifiers import ExtractIntermediateLayer,
    ReplaceToLinear
```

Listing 1: Example Lines Important Function Imports

3.2 Model Construction

The credit is mainly attributed to the creator of this kaggle notebook that I transferred to google colab at: `https://www.kaggle.com/code/kunjushaji/identification-of-plant-leaf-diseases-using-cnn`

The Model is a simple 6-layer Convolutional Neural Network (CNN) taking an input image of size 224x224x3, not using padding, and having a final conv filter with 512 filters of size 3x3. Each layer's convolution was a 3x3 filter, and the output layer are 2 dense (fully connected node [FCN]) that transforms to 1024 then to 38 or the required number of classes (changed to 61 for transfer learning dataset with different number of classes). Figure 7 shows the corresponding number of increasing filers, dense sizes, and image size as the values progress through the network and features are determined from different parts of the image.

Since no padding was used on the input layer, the input image size of 224x224 was reduced by 2 pixels in both dimensions due to the nature of the 3x3 Convolution Filter Size. Using a stried of 1 we can calculate the output from max-pooling0 after conv0 yields 111x111 we can determine the input to conv 1 by using:

Output Size = (Input Size - Filter Size) / Stride + 1 = (111 - 3) / 1 + 1 = 109

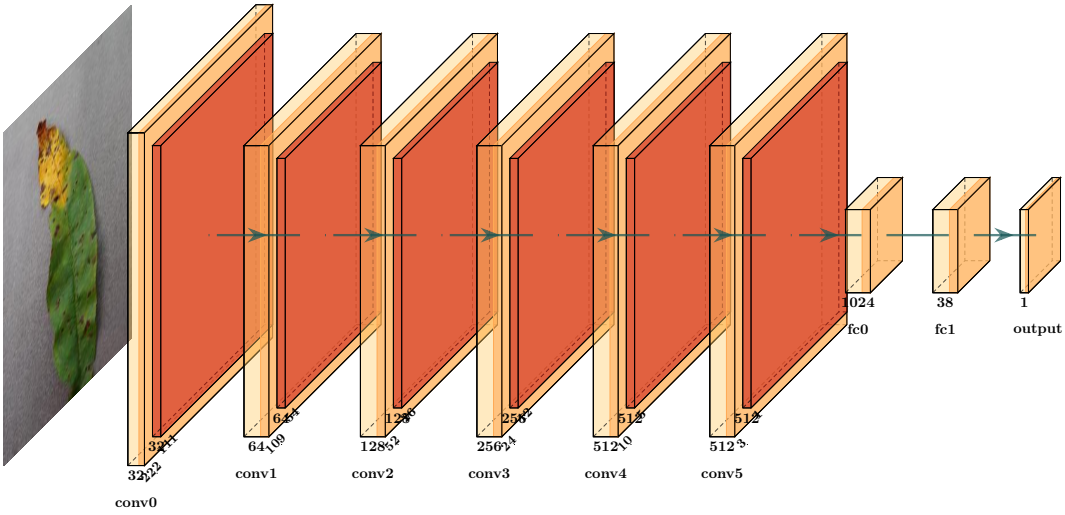


Figure 7: Convolutional Neural Network With Original Dataset - New Plant Disease Dataset with Augmentation

3.3 Data Display

Using Examples from `https://keisen.github.io/tf-keras-vis-docs/` I attempted to use Gradient Class Activation Map (GradCAM) but did not have much luck. I was just seeing no activations for any part of the image (blue/cool heatmap overlay on the leaf image). In general these saliency maps aim to define where a particular region is different/noticeable from its neighbors with respect to image features. I used a similar method called Maxmization Activation that works by randomly initializing the input image pixels to the specific filter layer chosen (conv 5). Then the **tf-keras-vis** *ActivationMaximization()* method is called to try and maximize the outputs of the current layer. These are then normalized and the strongly activated neurons give instight to learned features of the convolutional filters of that layer in the network. Note that the *ReplaceToLinear()* method is called as well to change the output softmax layer to be linearized in order to view the activation maximization. I also visualized the first layer in the network 6 of the 32 convolutional filters where black corresponds to 0 and white 1 for normalized filter weights.

Figure 8 shows the 1st layer normalized convolution filters 6/32 for that layer. Different filters have different normalized weighting. Figure 9 shows the Activation Maximization map showing important features for the last convolution layer (conv 5) in the network. Note that these are just 3 of the 512 size 3x3 filters in this layer. Finally, Figure 10 shows the attempt at the GradCAM map to display the relative linearized gradients to create a heatmap similar to that seen by the 3 convolution filters, but over the entire convolution (or dense or max-pooling if chosen to view) layer instead of just a select number of filters.

3.4 Original Network Performance

- The GPU in google colab performed around 135ms/step where 1 step is an iteration and there are 2812 iterations per epoch each having the set batch size.
- The Optimizer chosen was Adam with a Learnning Rate of 1e-4, Optimizing/Loss function of Categorical Cross Entropy with accuracy metric in model.compile.
- In the training 2812 is the number of iterations (how many batches are gone through to achieve # samples seen = 2812 x batch_size).

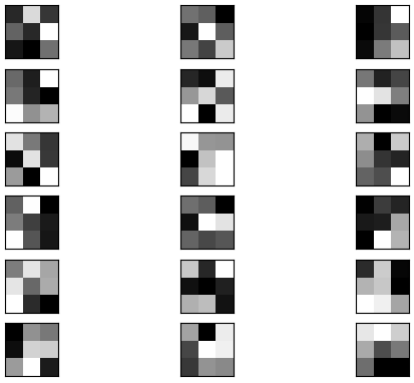


Figure 8: 1st layer of the CNN 6/32 normalized 3x3 filter weights

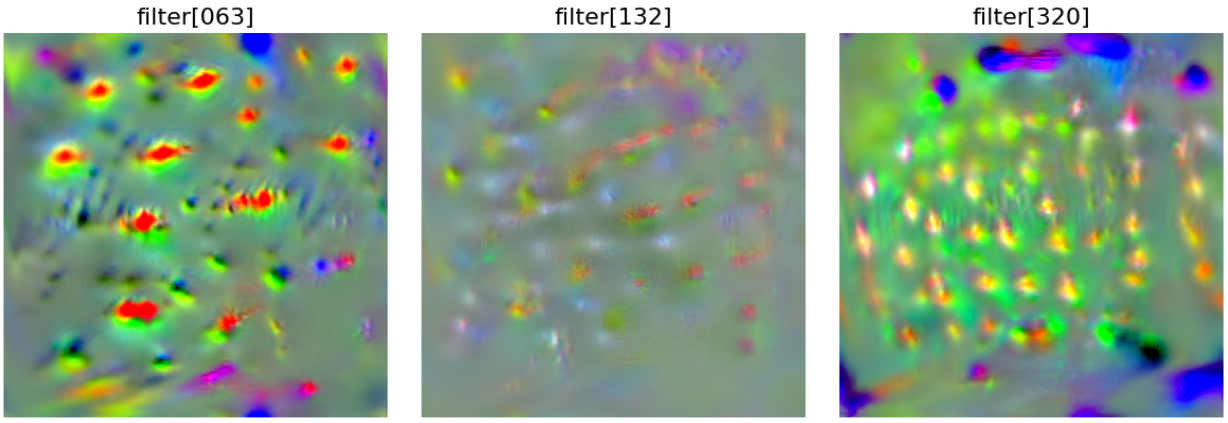


Figure 9: Convolution Outputs from 3 filters in the last convolution layer of the CNN

Figure 10: Attempted GradCAM heatmap where features are present, but not shown

- The number of trainable parameters for the original dataset are: 2,723,046 (10.39 MB) which is fairly small as typical CNNs are up to 30MB in size.
- Transfer Learning model: 2746621 (10.48 MB) more trainable parameters since output dense layer is of size 61 with more classes

The main conclusion from 3 is that the CNN performs better than classical methods, but transfer learning could use more Epochs or increased batch size because it seems to not be overfitting (reduced validation accuracy with increased training accuracy), but might be data starved.

The networks with higher epochs and batch sizes performed best with the Transfer learning network performing pretty well on just 5 epochs. The 12 epoch, 50 batch size accuracy/loss performance plot is in Figure 11. The same is done for the Transfer Learning network in Figure 12. Finally, Figure 13 displays the confusion matrix for the 12 epoch 25 batch size case. Confusion Matrices are important because they contain all (38) classes on the x and y axes and the diagonal represents correct classifications and it shows the number of classifications that were correct based on the "True Labels" on the y-axis and the "Predicted Labels" on the x-axis.

Table 3: Performance Comparison for different number epochs & Transfer Learning

# Epochs and Batch Size	**Model Size (#weights/$(1024)^2$)**	**Time for Validation**	**Training & Validation Accuracy**
6 Epochs 25 Batch Size	31.25 MB	Train = 200s per epoch 1200s	Train = Val = 0.983
12 Epochs 25 Batch Size	32.768MB	Train = 2120s (about 177s per epoch)	Train = 0.9804 Validate = 0.9832
12 Epochs 50 Batch Size	32.768MB	Train = 2108s	Train = 0.9805 Validate = 0.9777
5 Epochs 25 Batch Size **Transfer Learning**	32.275MB	Train = 914s	Train = 0.8572 Validate = 0.7933

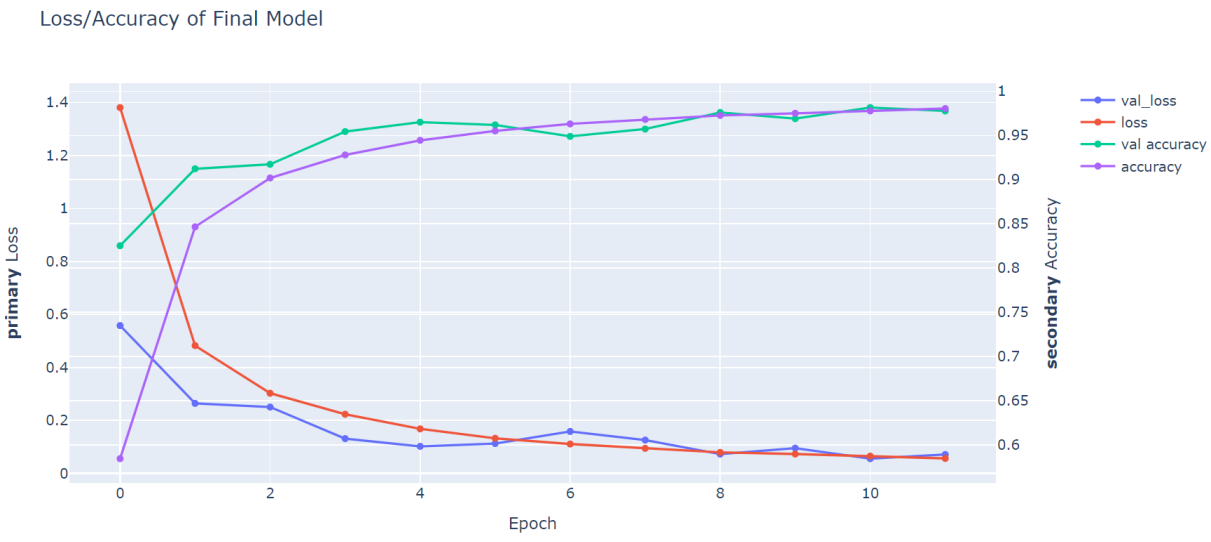


Figure 11: Performance Results Accuracy/Loss - Epoch = 12 Batch Size = 50

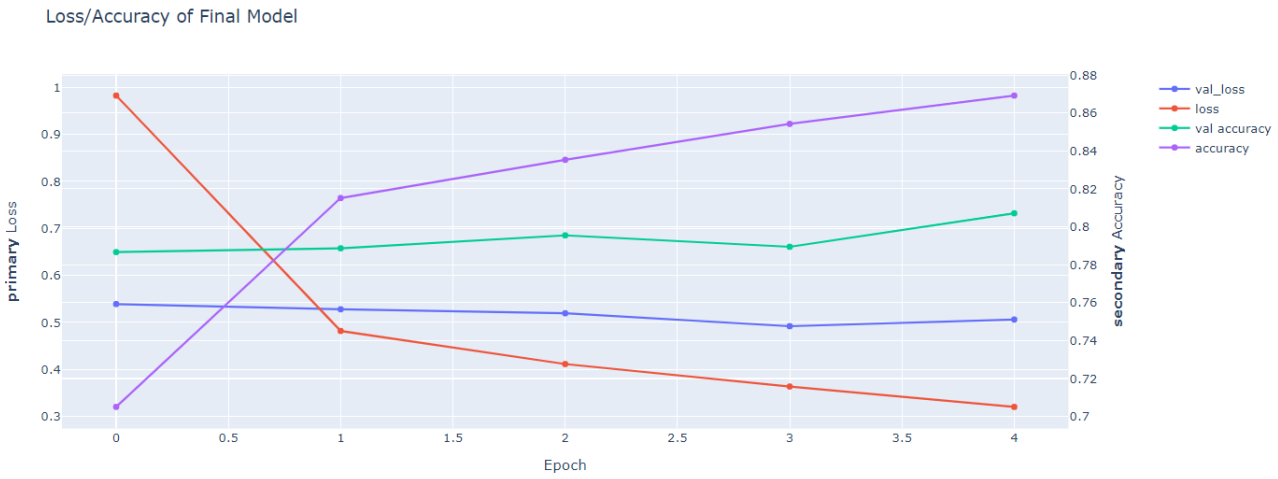


Figure 12: Performance Results Accuracy/Loss for Transfer Learning - Epoch = 5 Batch Size = 25

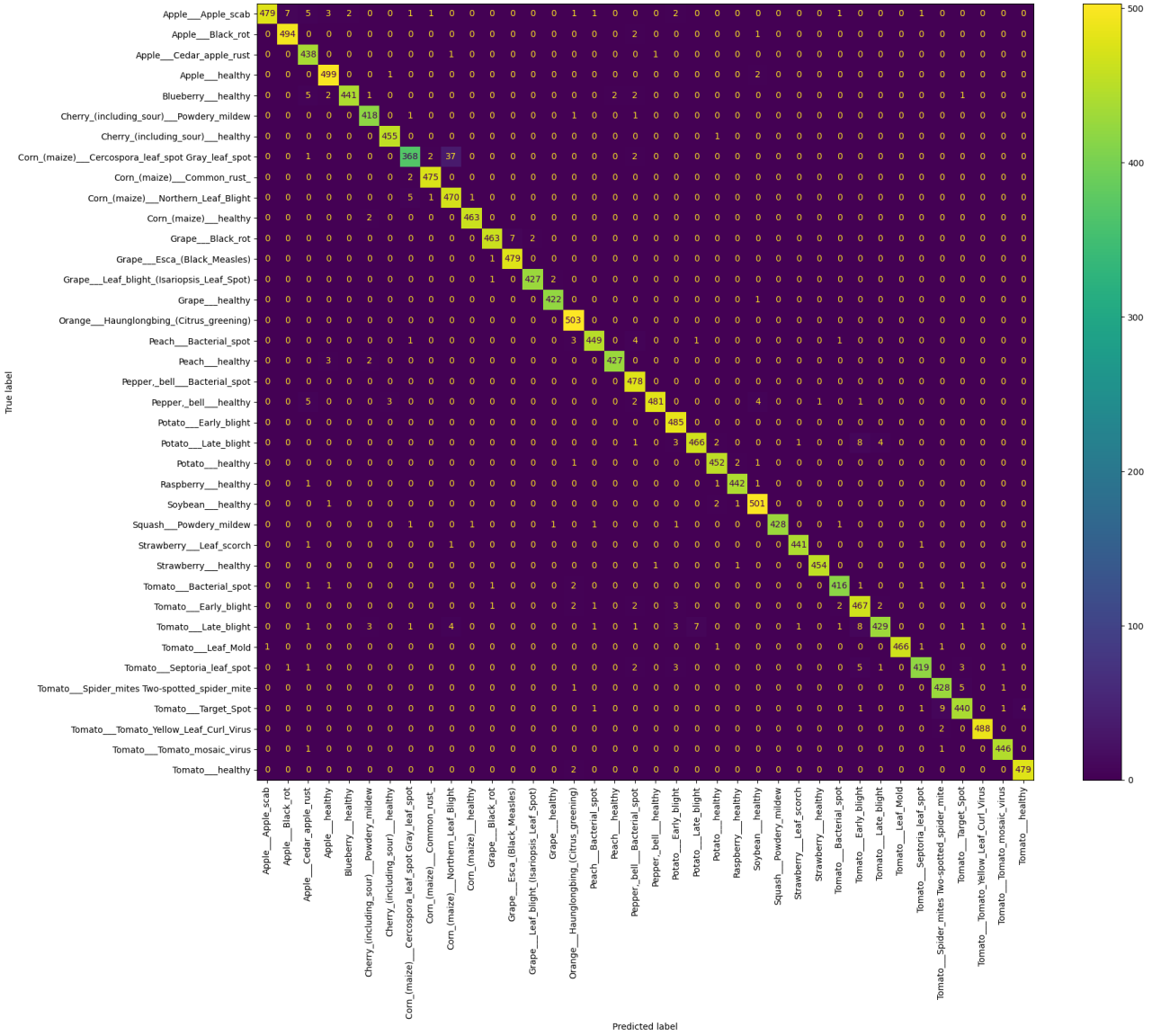


Figure 13: Confusion Matrix between 38 classes - Epoch = 12 Batch Size = 25

3.5 Transfer Learning

The main thing to note is the need to change the output layer from 38 to 61 to correspond to the number of classes in the new dataset. Doing this is shown below, where we initially use the trained model from the other dataset to initialize values. The new Dense output values will be randomized. *It would be an interesting experiment to try and place the dataset before any training and see the performance of the two datasets, but most likely there would be low accuracy, nonetheless it would be interesting to prove that as true.*

```
model.pop()  # Remove the last Dense layer

# Add a new Dense layer with output size 61
new_dense_layer = Dense(61, activation='softmax', name='dense_3_new')

# Add the new Dense layer to the model
model.add(new_dense_layer)

# Manually transfer weights from the old Dense layer to the new Dense layer
old_dense_weights = model.layers[-2].get_weights()

# Allow new dense weights to be initialized randomly -
# get_weights and set_weights will not work due to shape mismatch

# Compile the new adjusted model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['
    accuracy'])

# model.summary() should now list the last layer as: "dense_3_new (Dense) (
    None, 61) 62525"
```

Listing 2: Example Lines Important Function Imports

The training dataset was the New Plants Diseases Dataset from Kaggle discussed earlier. The example code used a MobileNet on the same data (which would have been a good comparison, or implementing a different network topology and pre-trained weights for transfer learning). For this project I used the AI Challenger 2018 Plant Disease Dataset found in [10]. In this paper, when categories had less than 300 images, plant disease images were scraped from the internet to bolster the number and synthetic generate images were produced with the keras functionality to transform and increase the dataset size via rotate images and applying other geometrical transforms (affine, brightness shifts).

Since the data was not labeled (did not have the key values) the paper took the figure put it into an OCR software, added quotes in Excel and create dictionary quickly. Copy the values in and adjust them (some general/serious might get out of order key value pairs for the dictionary). In excel create a third column where the column values are: **= TEXT(B2,"\'@\'")&":"&A2** have to use single ' since excel uses " for maintaining the format and this will just need to be modified with commas in a word editor and can be used.

To download the data into google colab, the opendatasets library was used. This worked easier than the kaggle implementation as long as you create a kaggle API key and use the files.upload() method to put it into to your workspace.

4 Conclusion

Overall this project taught me how to visualize convolutional filters and the potential for GradCAM, the importance of model size (number of trainable parameters), The performance improvements of Neural Networks compared to classical methods (SVMs for multi-categorical classification problems), and the wide variety of networks available to be used with transfer learning.

One major frustration was the use of google colab. While the resource is great for training networks using GPUs, it was challenging to have to re-run everything (not stored in the workspace) when leaving / disconnecting. Furthermore, GPUs would sometimes crash even if you were below the system's RAM usage allowed. Running through each step to create functions and download data to the cloud machine. On the other hand, loading models was not complicated and the interface is very straightforward most of the time.

I really enjoyed learning about neural networks for this project, but wish I had more time to improve on a few things. More time to explore and access other research, comparison against classical methods like K-nearest neighbors or Support Vector Machines, along with more time to understand Transformers and Generative adversarial networks used in some papers I came across. Another promising looking field is that of self-attention neural networks. Below I describe some of the more reasonable future goals for this project.

Future Improvements:

- Future goals would be to determine the type of leaf then if multiple deases are present draw bounding boxes and lable each disease on the leaf similar to what's done here (but only for one disease) using something like Masked RCNN like in Figure 14.
- Another interesting task would have been to create a dataset by scraping the internet for images pertaining to a specific disease class using something like `https://www.scraperapi.com/blog/how-to-build-your-own-dataset-with-web-scraping/`
- Using datasets like PlantDoc and FieldPlant would have been nice to compare performance of plants in realistic environments instead of lab-environment training images

Figure 14: Grape Measles Showing each region where the leaf has the occurrence of the disease Source: `https://github.com/AarohiSingla/Plant-Disease-Detection-Using-Mask-R-CNN/tree/main`

References

[1] P. S. Thakur, T. Sheorey, and A. Ojha, "Vgg-icnn: A lightweight cnn model for crop disease identification," *Multimedia Tools and Applications*, vol. 82, pp. 497–520, 1 2023.

[2] E. Moupojou, A. Tagne, F. Retraint, A. Tadonkemwa, D. Wilfried, H. Tapamo, and M. Nkenlifack, "Fieldplant: A dataset of field plant images for plant disease detection and classification with deep learning," *IEEE Access*, vol. 11, pp. 35398–35410, 2023.

[3] X. Dong, Q. Wang, Q. Huang, Q. Ge, K. Zhao, X. Wu, X. Wu, L. Lei, and G. Hao, "Pddd-pretrain: A series of commonly used pre-trained models support image-based plant disease diagnosis," *Plant Phenomics*, vol. 5, 2023.

[4] J. Andrew, J. Eunice, D. E. Popescu, M. K. Chowdary, and J. Hemanth, "Deep learning-based leaf disease detection in crops using images for agricultural applications," *Agronomy*, vol. 12, 10 2022.

[5] A. K. Somani, A. Mundra, R. Doss, and S. Bhattacharya, "Computing proceedings of ssic 2021."

[6] S. Jadon, "Ssm-net for plants disease identification in low data regime," 5 2020.

[7] Q. H. Cap, H. Uga, S. Kagiwada, and H. Iyatomi, "Leafgan: An effective data augmentation method for practical plant disease diagnosis," 2 2020.

[8] P. S. Thakur, P. Khanna, T. Sheorey, and A. Ojha, "Explainable vision transformer enabled convolutional neural network for plant disease identification: Plantxvit," 7 2022.

[9] W. Zeng and M. Li, "Crop leaf disease recognition based on self-attention convolutional neural network," *Computers and Electronics in Agriculture*, vol. 172, 5 2020.

[10] Y. Liu, G. Gao, and Z. Zhang, "Crop disease recognition based on modified light-weight cnn with attention mechanism," *IEEE Access*, vol. 10, pp. 112066–112075, 2022.