VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

**Programming Integration Project (CO3101)**

---

**CC06 - Technical Report**

# RGB Image-Based Disease Detection in Plants with Computer Vision

---

| | | |
|---|---|---|
| Advisor(s): | Nguyen An Khuong | |
| Student(s): | Nguyen Van An | 2352013 |
| | Hoang Kim Cuong | 2352145 |

HO CHI MINH CITY, DECEMBER 2025

## Member list & Workload

| No. | Full name | Student ID | Task |
|---|---|---|---|
| 1 | Nguyen Van An | 2352013 | Dataset preprocessing, and training of multiple CNN models (ResNet-18, MobileNetV1, MobileNetV2, VGG-16). |
| 2 | Hoang Kim Cuong | 2352145 | Report writing, documentation, result analysis, and training of one baseline model (ResNet-18) |

## Video presentation

Our group's presentation can be viewed here
The slides we used for our presentation can be viewed here

# Contents

# List of Figures

## 1 Introduction

In this project, we focus on RGB image-based disease detection in plants using deep learning techniques. Four CNN architectures are implemented and evaluated in this study, including **ResNet-18, MobileNetV1, MobileNetV2, and VGG-16**. These models represent different design philosophies in terms of depth, computational complexity, and efficiency. After training, the models are compared based on multiple evaluation criteria.

The main objective of this project is to identify the most suitable deep learning model for RGB-based plant disease detection by analyzing the trade-offs between performance and computational efficiency. The experimental results provide valuable insights for developing practical, real-world plant disease detection systems, especially for deployment on resource-constrained devices.

## 2 Dataset Description

The dataset used in this project is the PlantVillage dataset, obtained from Kaggle under the identifier **abdallahalidev/plantvillage-dataset**. PlantVillage is one of the most widely used public benchmark datasets for plant disease detection using computer vision. The dataset was created through a large-scale data collection effort involving agricultural experts and researchers. Leaf images were captured under controlled laboratory conditions with uniform background, consistent lighting, and fixed camera distance to minimize environmental noise.

The full dataset contains more than 50,000 leaf images covering multiple crop species and disease categories, including both healthy and infected samples. Each image represents a single leaf with clearly visible disease patterns such as spots, blights, rust, and discoloration. Because of its size, quality, and standardized structure, PlantVillage is well suited for training and benchmarking deep learning models.

The Kaggle version of the dataset is organized into three main subsets: Color (RGB) images, Grayscale images and Segmented images. Since this project focuses on RGB-based plant disease detection, only the **color image subset** is used throughout all experiments.

## 3 Dataset Preprocessing

The dataset is subsequently filtered to include only six target crop species, namely **Apple, Corn, Grape, Peach, Potato, and Strawberry**, while all remaining plant categories are excluded. For each preserved plant–disease class, the corresponding images are randomly shuffled to mitigate potential ordering bias and to promote statistical representativeness.

The shuffled data are then partitioned into three mutually exclusive subsets according to a **70%–15%–15% split,** corresponding to the training, validation, and test sets, respectively to ensure a proper balance between learning capability, model tuning, and unbiased evaluation. **The training set (70%)** provides sufficient data for the deep learning models to effectively learn complex visual patterns of plant diseases. **The validation set (15%)** is used during training to tune hyperparameters, monitor overfitting, and select the best-performing model without influencing the final results. **The test set (15%)** is kept completely independent and is only used for the final performance evaluation, ensuring that the reported accuracy reflects the true generalization ability of the model. This split ratio is widely adopted in computer vision research because it offers a good trade-off between training data sufficiency and evaluation reliability, especially for medium-to-large datasets like **PlantVillage.**

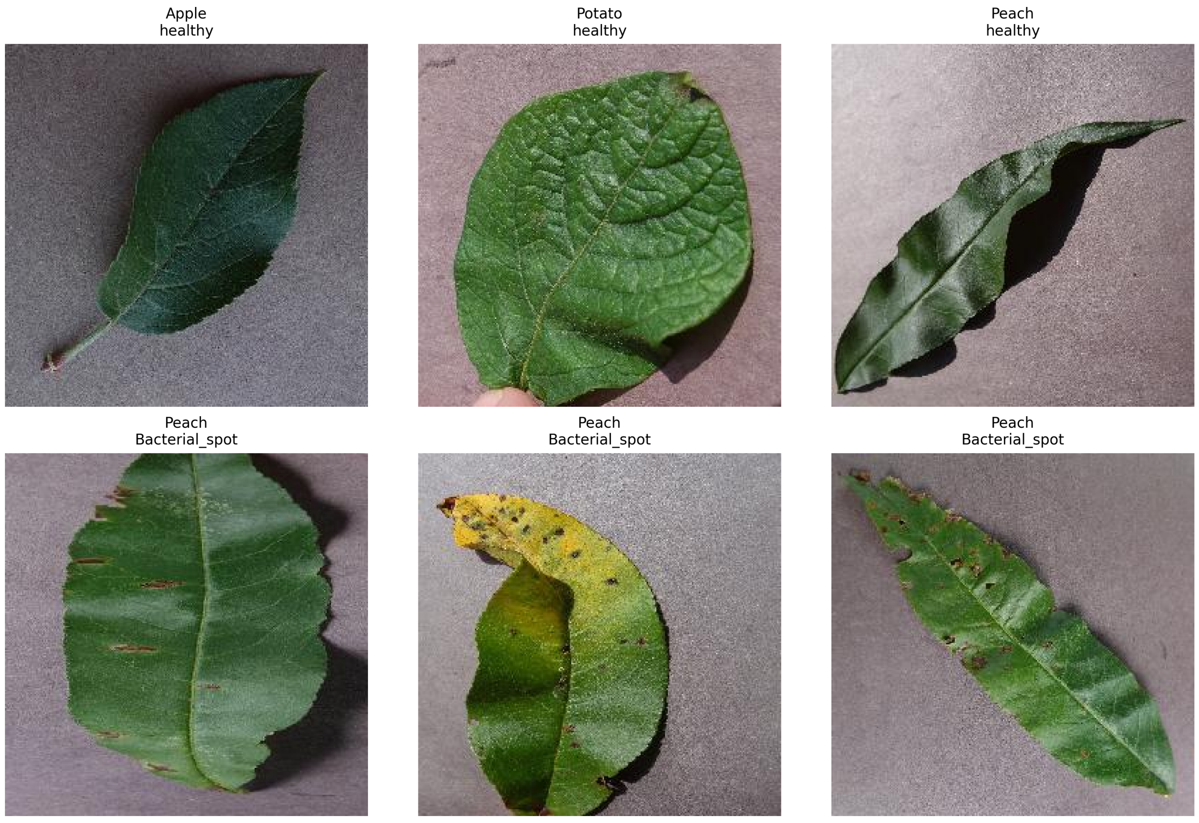
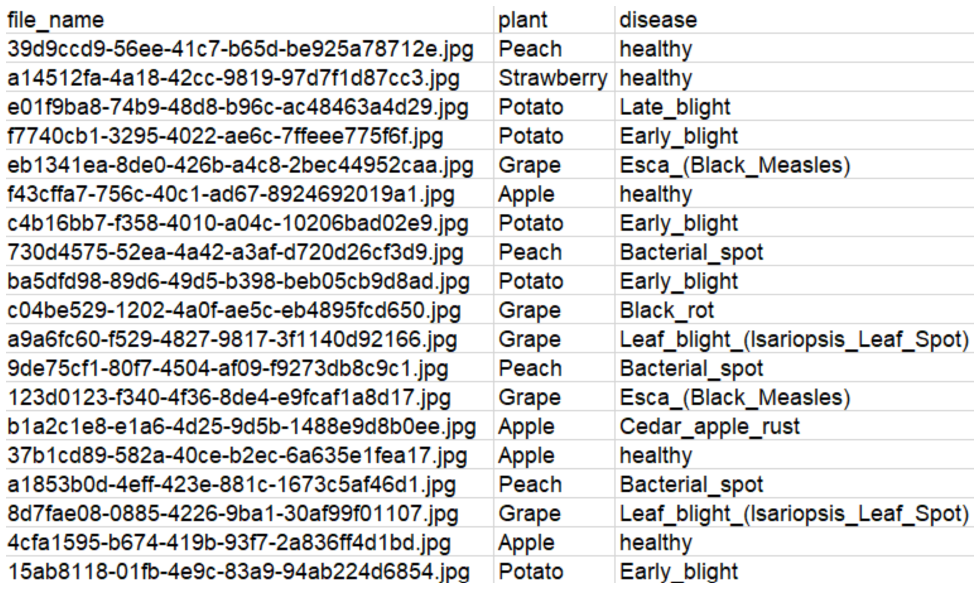


Figure 3.1: Train samples

The dataset labels are stored in CSV files **(train_labels.csv, val_labels.csv, and test_labels.csv)** to provide a clear, structured, and reproducible mapping between images and their corresponding annotations. Each CSV file contains three essential fields: image filename, plant category, and disease class, which directly supports supervised learning. This labeling format allows the dataset to be easily loaded using standard machine learning tools such as **Pandas, PyTorch, and Scikit-learn**, without relying on rigid folder structures. Moreover, storing the data split in separate CSV files ensures that the exact same train–validation–test partition can be reused across multiple experiments, which is critical for reproducibility, fair model comparison, and statistical analysis. This design also enables flexible data filtering, class distribution analysis, and performance evaluation without physically reorganizing the image files.



| file_name | plant | disease |
|---|---|---|
| 39d9ccd9-56ee-41c7-b65d-be925a78712e.jpg | Peach | healthy |
| a14512fa-4a18-42cc-9819-97d7f1d87cc3.jpg | Strawberry | healthy |
| e01f9ba8-74b9-48d8-b96c-ac48463a4d29.jpg | Potato | Late_blight |
| f7740cb1-3295-4022-ae6c-7ffeee775f6f.jpg | Potato | Early_blight |
| eb1341ea-8de0-426b-a4c8-2bec44952caa.jpg | Grape | Esca_(Black_Measles) |
| f43cffa7-756c-40c1-ad67-8924692019a1.jpg | Apple | healthy |
| c4b16bb7-f358-4010-a04c-10206bad02e9.jpg | Potato | Early_blight |
| 730d4575-52ea-4a42-a3af-d720d26cf3d9.jpg | Peach | Bacterial_spot |
| ba5dfd98-89d6-49d5-b398-beb05cb9d8ad.jpg | Potato | Early_blight |
| c04be529-1202-4a0f-ae5c-eb4895fcd650.jpg | Grape | Black_rot |
| a9a6fc60-f529-4827-9817-3f1140d92166.jpg | Grape | Leaf_blight_(Isariopsis_Leaf_Spot) |
| 9de75cf1-80f7-4504-af09-f9273db8c9c1.jpg | Peach | Bacterial_spot |
| 123d0123-f340-4f36-8de4-e9fcaf1a8d17.jpg | Grape | Esca_(Black_Measles) |
| b1a2c1e8-e1a6-4d25-9d5b-1488e9d8b0ee.jpg | Apple | Cedar_apple_rust |
| 37b1cd89-582a-40ce-b2ec-6a635e1fea17.jpg | Apple | healthy |
| a1853b0d-4eff-423e-881c-1673c5af46d1.jpg | Peach | Bacterial_spot |
| 8d7fae08-0885-4226-9ba1-30af99f01107.jpg | Grape | Leaf_blight_(Isariopsis_Leaf_Spot) |
| 4cfa1595-b674-419b-93f7-2a836ff4d1bd.jpg | Apple | healthy |
| 15ab8118-01fb-4e9c-83a9-94ab224d6854.jpg | Potato | Early_blight |

Figure 3.2: CSV files

This comprehensive preprocessing strategy guarantees that the curated dataset is well-structured, reproducible, and fully compatible with modern deep learning training and evaluation pipelines. Furthermore, by enforcing a standardized directory hierarchy, fixed random splitting, and explicit CSV-based annotation, the entire data preparation process becomes transparent, traceable, and repeatable across multiple experimental settings. This structured design not only minimizes the risk of data leakage between training, validation, and test sets, but also ensures that all evaluated models are trained and tested under identical and fair conditions. As a result, the proposed preprocessing pipeline provides a reliable foundation for objective model comparison, statistical performance analysis, and future extensions of the dataset or experimental framework.

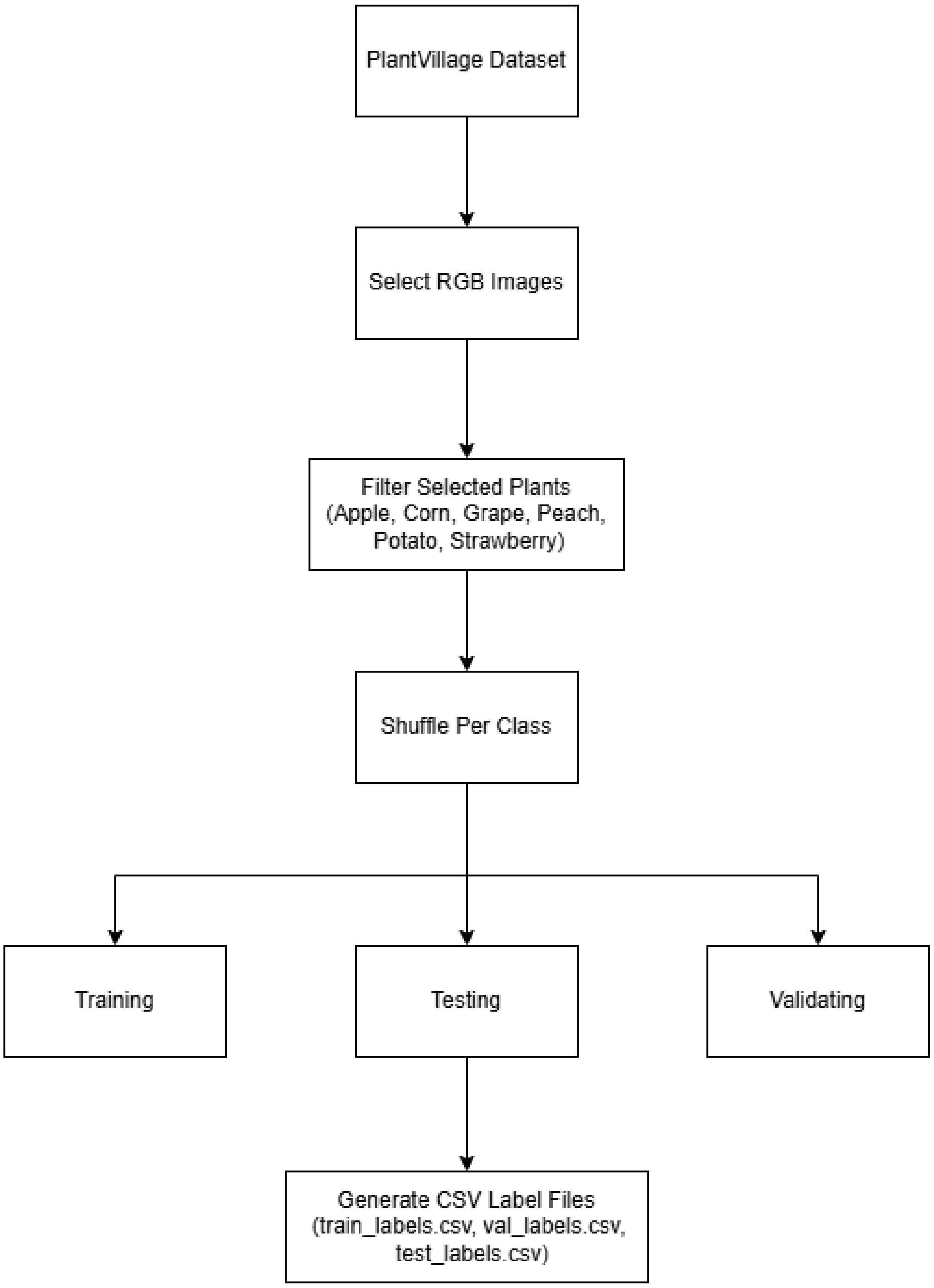

PlantVillage Dataset
Select RGB Images
Filter Selected Plants (Apple, Corn, Grape, Peach, Potato, Strawberry)
Shuffle Per Class
Training
Testing
Validating
Generate CSV Label Files (train_labels.csv, val_labels.csv, test_labels.csv)

Figure 3.3: Data preprocessing pipeline

# 4 Model Training

In this section we will give detailed descriptions and architectures of our candidate models, which we chose based on Simplicity, Efficiency, and Size.

## 4.1 Model Architecture

### 4.1.1 VGG-16 Architecture

VGG-16 is a deep convolutional neural network composed of 16 trainable layers, including 13 convolutional layers and 3 fully connected layers. The network follows a strictly sequential and uniform architecture, in which all convolutional layers use 3×3 kernels with stride 1 and padding 1, while 2×2 max-pooling layers with stride 2 are applied after each convolutional block for spatial downsampling.

The architecture is organized into five convolutional blocks. The first two blocks contain two convolutional layers each, while the last three blocks contain three convolutional layers each. After the convolutional feature extraction stage, the output feature maps are flattened and passed through three fully connected layers, where the first two layers contain 4096 neurons each, and the final layer outputs the class probabilities via a Softmax activation function.

This is the most simple and straightforward architecture. However, due to it's simplicity, it has to compensate with size. Indeed, a rough estimation showed that the model has around 102 million parameters, which is larger than all of the other candidate models combined.

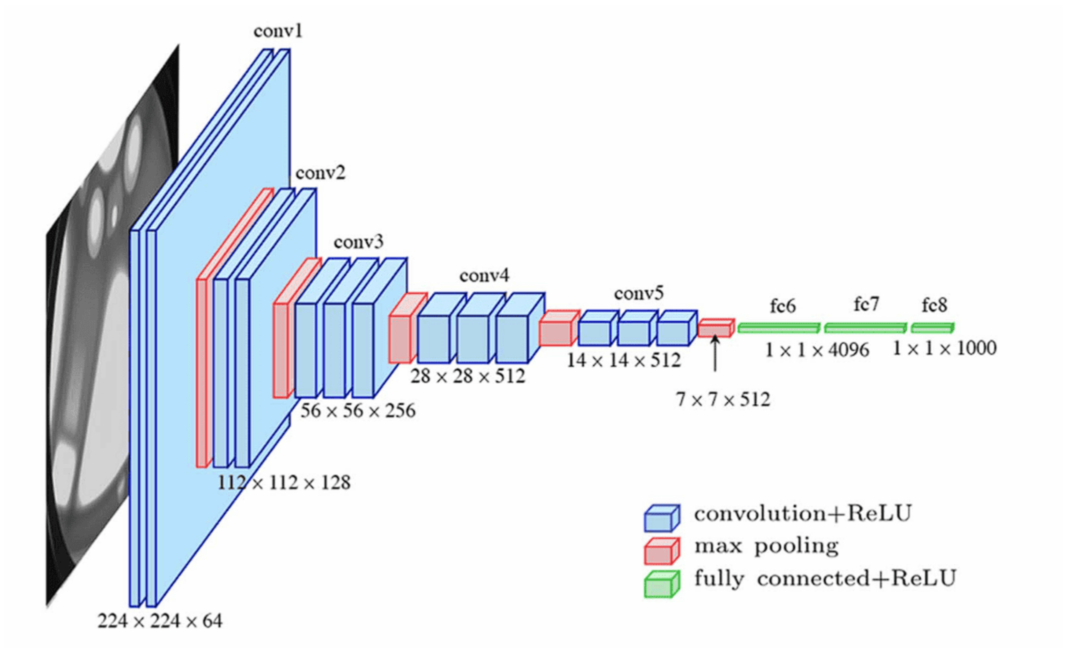


Figure 4.1: VGG-16 Architecture

### 4.1.2 ResNet-18 Architecture

ResNet-18 is a residual convolutional neural network consisting of 18 layers, including convolutional, pooling, and fully connected layers. The defining characteristic of ResNet-18 is the use of residual blocks with identity shortcut connections, which enable direct information flow across layers and effectively mitigate the vanishing gradient problem.

Each residual block consists of two 3×3 convolutional layers, followed by batch normalization and ReLU activation. The input to the block is added directly to the output via a skip connection, allowing the network to learn residual mappings instead of direct feature transformations.

The overall architecture begins with a 7×7 convolutional layer followed by max-pooling, and is then divided into four residual stages with increasing channel depth (64, 128, 256, and 512). Global average pooling is applied before the final fully connected layer to reduce overfitting and model complexity.

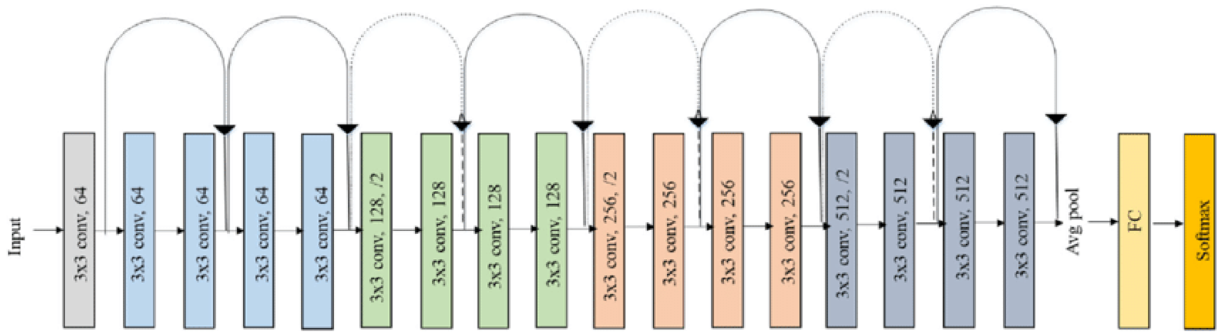


Figure 4.2: ResNet-18 Architecture

### 4.1.3 MobileNetV1 Architecture

MobileNetV1 is a lightweight convolutional neural network specifically designed for mobile and embedded vision applications. Its core architectural innovation is the use of depthwise separable convolution, which decomposes a standard convolution into two consecutive operations:

- Depthwise convolution, applied independently to each input channel for spatial feature extraction.
- Pointwise convolution (1×1 convolution), used for channel-wise feature combination.

This factorization drastically reduces the number of parameters and computational cost compared to standard convolutions. The network is constructed from a sequence of depthwise separable convolution blocks, each followed by batch normalization and ReLU activation. Downsampling is performed by increasing the convolution stride at specific layers.

Despite its compact structure and high inference speed, MobileNetV1 is outdated and has been superseded by a lighter, faster and better overall MobileNetV2.

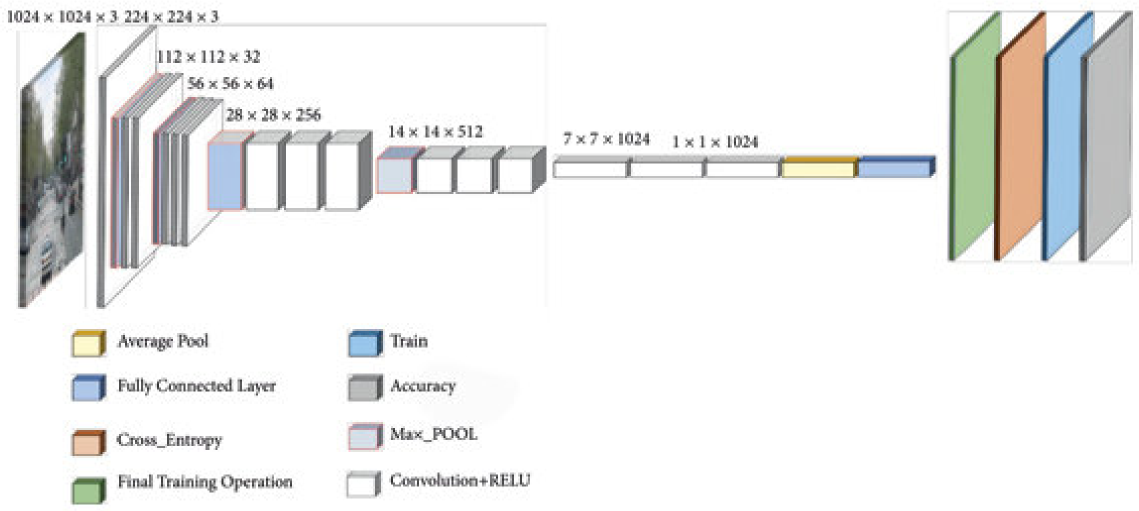


Figure 4.3: MobileNetV1 Architecture

### 4.1.4 MobileNetV2 Architecture

MobileNetV2 is an enhanced version of MobileNetV1 and introduces two major architectural improvements:

- Inverted residual blocks: Unlike traditional residual blocks that compress features inside the block, MobileNetV2 uses an inverted structure. The input is first expanded to a higher dimension using a $1 \times 1$ convolution, processed by a $3 \times 3$ depthwise convolution, and finally projected back to a low-dimensional space using a $1 \times 1$ convolution. Crucially, the shortcut connection is applied between the narrow bottleneck layers (input and output) rather than the expanded layers.

- Linear bottlenecks: The MobileNetV2 authors observed that applying non-linear activation functions (like ReLU) in low-dimensional spaces destroys information. [5] To mitigate this, the non-linearity is removed from the final $1 \times 1$ projection layer of the block, leaving it as a linear bottleneck. This preserves the information compressed in the lower-dimensional output.

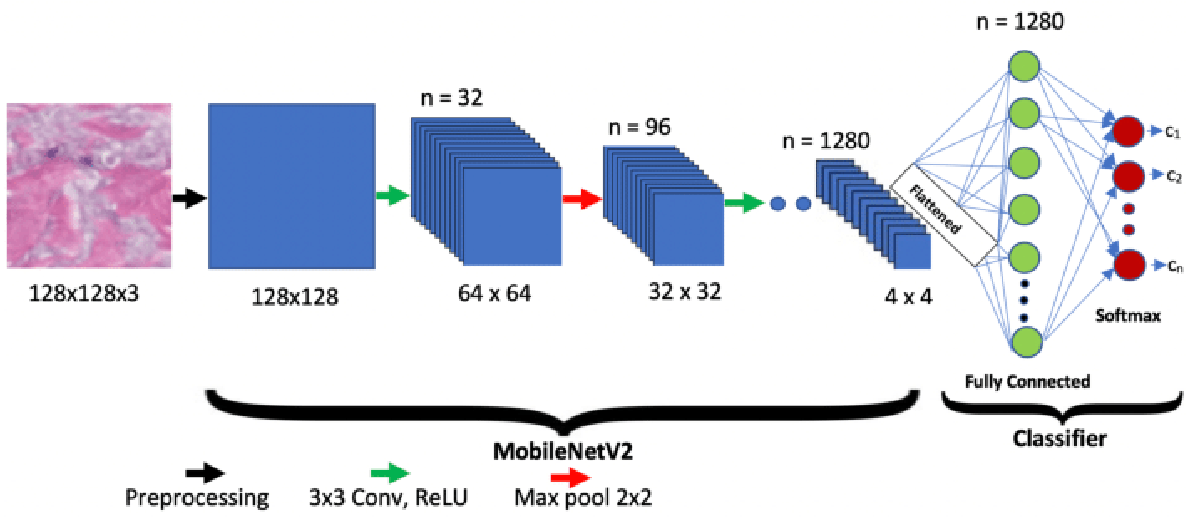


Figure 4.4: MobileNetV2 Architecture

## 4.2 Data Preparation

- **Normalization:** Input normalization is crucial because it helps neural networks converge faster, remain stable, and reduces the risk of vanishing or exploding gradients. For images, pixel values originally range from [0, 255] and are usually scaled to [0, 1] using ToTensor(). However, without further normalization, these values can still be large, causing skewed activations, high variance, and unstable gradients. To address this, we normalize the images by subtracting the mean and dividing by the standard deviation. The mean and standard deviation for the three color channels (R, G, B) are [0.485, 0.456, 0.406] and [0.229, 0.224, 0.225], respectively. These values were taken from ImageNet's statistics as an approximation. This normalization roughly scales pixel values to the [-1, 1] range, centers the data around zero, and makes it symmetric, which helps gradients remain balanced, optimizers work more effectively, and speeds up convergence during training.

$$x_{\text{norm}} = \frac{x - \mu}{\sigma}$$

  where:

  - $x$ is the original pixel value,
  - $\mu$ is the mean of the dataset (or the channel),
  - $\sigma$ is the standard deviation of the dataset (or the channel),

  - $x_{\text{norm}}$ is the normalized pixel value.

- **Data Augmentation:** Techniques like random rotations, horizontal flips, and shifts are applied to the training set to artificially increase its size and diversity, which helps the model generalize better and prevents overfitting. We divide the training dataset into 3 different augmentation levels: 60% of them use mild augmentation, 30% of them use moderate augmentation and 10% of them use aggressive augmentation.

| Mode | Applied Transformations | Objective |
|---|---|---|
| Mild Augmentation | Horizontal Flip and Vertical Flip | Increase symmetry and slightly expand dataset size, helping the model learn features that are not dependent on object orientation. |
| Moderate Augmentation | Flips + Random Rotations + Random Crops + Color Jitter | Simulate changes in camera angle (rotation, cropping) and lighting conditions (color variations) in real scenarios. Helps the model recognize objects even when position and color change slightly. |
| Aggressive Augmentation | Flips + Rotations + Crops + Color Jitter + Gaussian Blur | Create maximum diversity, simulating suboptimal photo conditions such as blurriness due to camera shake or lack of sharpness. Often used when the model tends to overfit severely or the original dataset is small. |

Table 4.1: Summary of data augmentation strategies and their objectives.

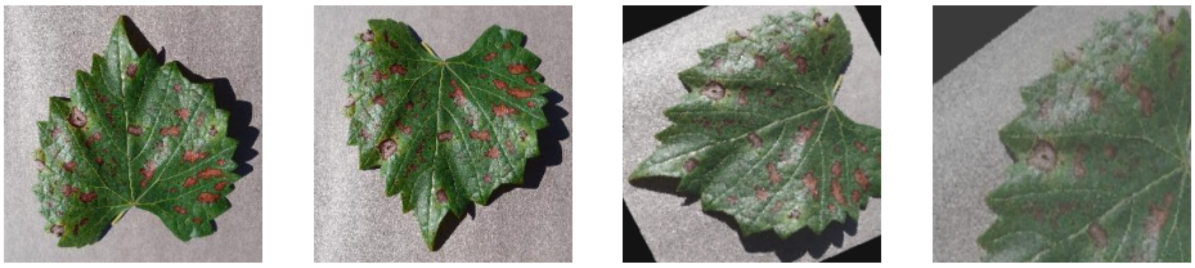


Figure 4.5: Augmentation of a Grape leaf with Esca. From left to right: Original, Mild, Moderate, and Aggressive Augmentations.

## 4.3 Training Process

The core of training a deep learning model is the iterative process of minimizing a loss function using an optimizer. In each iteration, input data passes through the network in a forward pass to generate predictions. The loss function then measures the difference between the predictions and the true labels. This error is propagated backward through the network during backpropagation, calculating gradients that indicate how much each weight contributed to the error. The optimizer uses these gradients to update the model's weights, with the learning rate controlling the step size of these updates. This process is repeated for many iterations, gradually refining the model parameters until the loss is minimized and the network achieves optimal performance.

# 5 Training and Evaluation

## 5.1 Metrics

To comprehensively assess the performance of the proposed plant disease classification models, **Accuracy** and **F1-score** are employed as evaluation metrics. While **Accuracy** provides a general overview of model correctness, it is insufficient on its own for imbalanced and high-risk classification problems such as plant disease detection. Therefore, **F1-score** are additionally reported to ensure a more reliable and interpretable evaluation.

## 5.2 Initialization

Since all models were trained from scratch, we used weight initializations to ensure the signal propagated effectively through the deep networks during the early stages of training. Specifically, we used the LeCun initialization for the fully connected (linear) layers and the Kaiming (also known as He) initialization for the convolutional layers.

The purpose of weight initialization was to prevent the gradients from vanishing or exploding during backpropagation, which is critical for convergence in deep architectures using ReLU-based activation functions.

## 5.3 Model Training

**ResNet18, MobileNetV1, MobileNetV2** were evaluated on the same test set using Accuracy and F1-score as the primary performance metrics. Since there is no significant difference in training speed among the three models, the comparison focuses on classification performance and model efficiency (size and number of parameters).

**VGG-16** was not feasible to train in this experiment due to its extremely high memory consumption. It requires significantly more GPU memory for storing weights, feature maps, and gradients during backpropagation. As a result, the model caused out-of-memory (OOM) errors on the available hardware.

The models were trained with the following parameters: `batch_size` $= 32$, `lr` $= 0.0002$, and `weight_decay` $= 0.001$. **ResNet18** trained for 10 epochs with patience 3, while **MobileNetV1/2** were trained for 15 epochs with patience 5. Furthermore, to align with PyTorch's implementation, we have also employed a Dropout layer with probability 0.2 to **MobileNetV2**.

## 5.4 Results

Below are the results that we collected from the training of models.

| Model | Accuracy (%) | F1-score | Model Size (MB) | Parameters (M) |
|---|---|---|---|---|
| ResNet18 | **97.23** | **0.9725** | 11.19 | 42.71 |
| MobileNetV1 | 93.01 | 0.9295 | 3.23 | 12.39 |
| MobileNetV2 | 94.78 | 0.9471 | **2.25** | **8.71** |

Table 5.1: Comparison of Model Performance on the Test Set

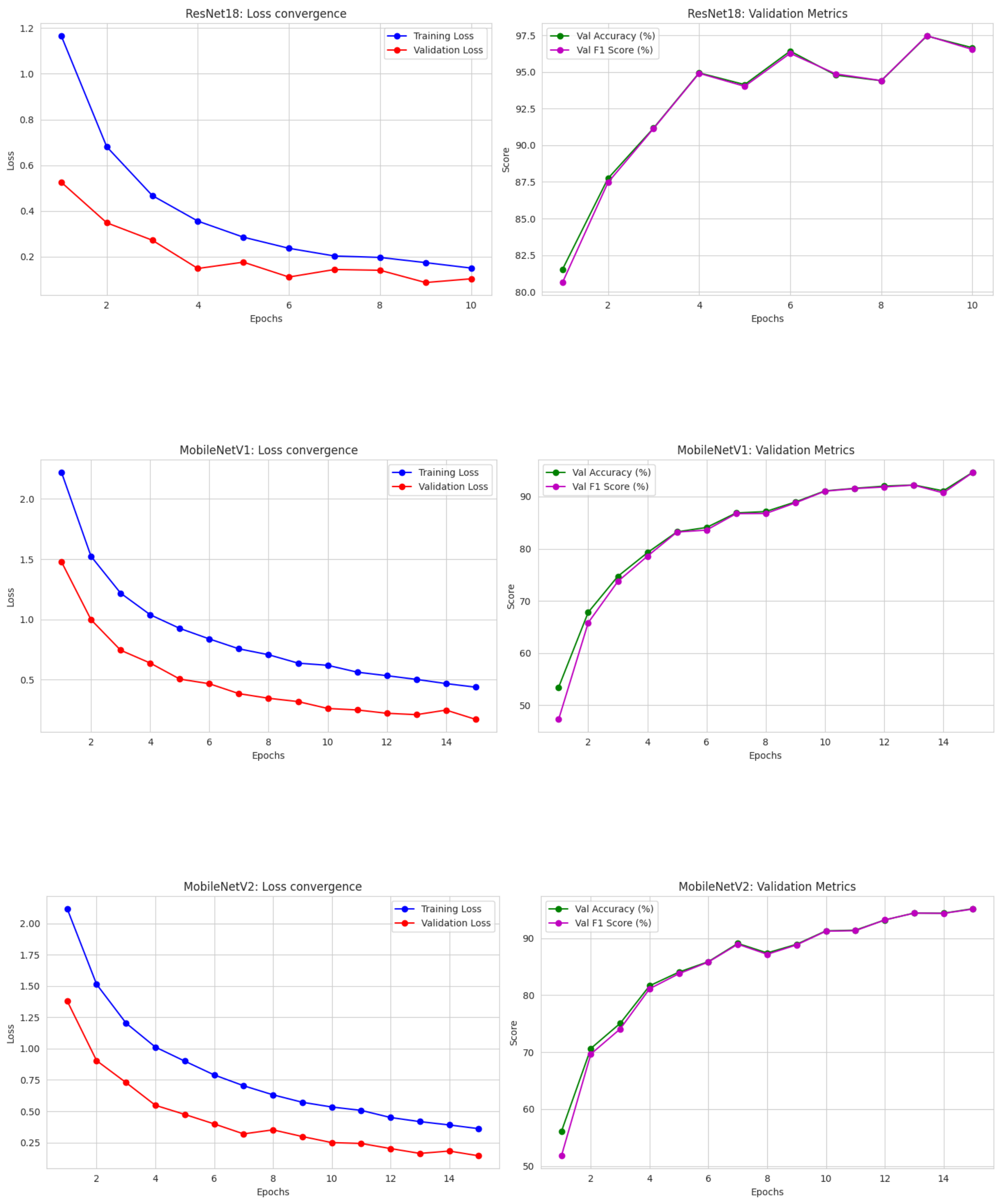
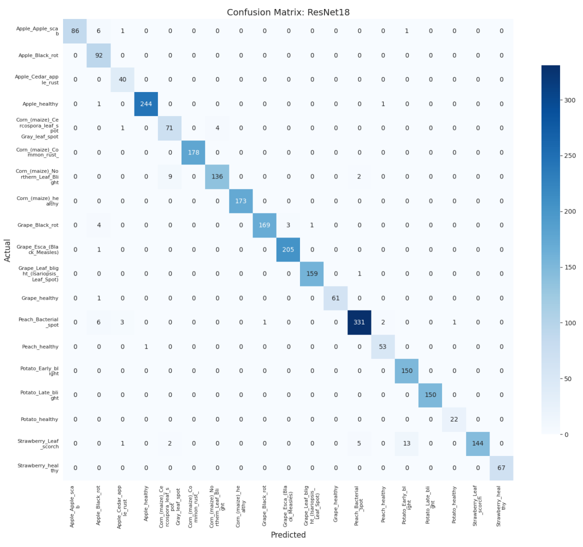
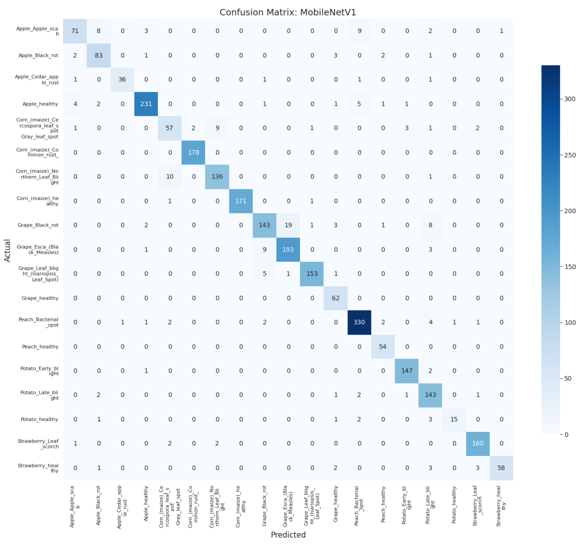


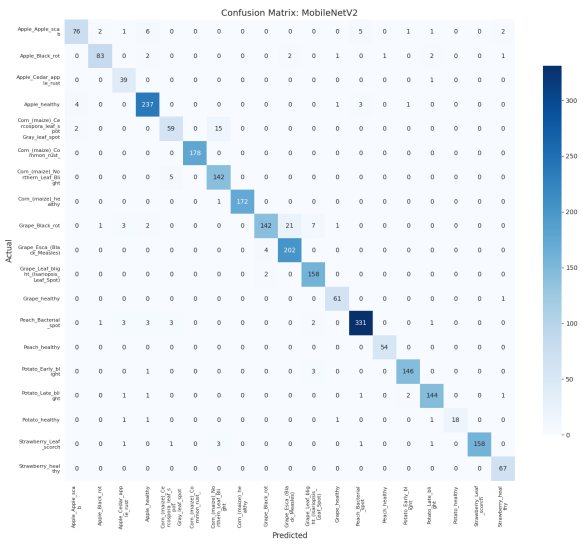
Figure 5.1: Comparison of training loss convergence and validation metrics for ResNet18, MobileNetV1, and MobileNetV2 over training epochs.



(a) ResNet18



(b) MobileNetV1



(c) MobileNetV2

Figure 5.2: Confusion matrices for all three models. Note how MobileNetV2 achieves similar classification density to ResNet18.

## 6 Discussion

Among the evaluated models, **ResNet18** achieved the best performance, with a accuracy of 97.23% and a F1-score of 0.9725, demonstrating very strong and balanced classification capability across all disease classes. It is also the largest model with the highest number of parameters (42.71 million), which contributes to its superior accuracy but results in higher memory consumption. **MobileNetV2** ranked second, achieving a accuracy of 94.78% and an F1-score of 0.9471, offering competitive performance while being the most lightweight model with only 2.25 MB and 8.71 million parameters, making it well-suited for deployment on mobile or edge devices. **MobileNetV1** obtained the lowest performance, with a accuracy of 93.01% and an F1-score of 0.9295, and although it is larger than **MobileNetV2** (3.23 MB, 12.39 million parameters), it is less optimal in terms of both accuracy and efficiency.

## 7 Conclusion

In this study, **ResNet18** was selected as the final model due to its superior classification performance on the test set. Although MobileNet architectures are more lightweight and computationally efficient, they exhibited noticeably lower accuracy. Specifically, **ResNet18** achieved a accuracy of **97.23%** and F1-score of **0.9725**, which are significantly higher than those of **MobileNetV1** and **MobileNetV2**. While **ResNet18** requires more parameters and higher memory consumption, the substantial improvement in detection accuracy justifies this trade-off. Therefore, **ResNet18** is chosen as the final model to ensure the highest reliability in plant disease detection.

## 8 Code and Data Availability

The source code for the Project as well as most figures can be accessed here.

The curated dataset which was used for training can be accessed here.

## 9 Summary

### 9.1 Project Analysis

#### 9.1.1 Strengths

A key strength of this project is that all tested models achieved relatively high performance, demonstrating that deep learning can effectively classify plant diseases from leaf images. Among them, **ResNet18** reached the highest accuracy and F1-score, providing very reliable predictions. **MobileNetV2** also performed well while being lightweight, and **MobileNetV1** achieved reasonable accuracy, showing that even smaller or simpler models can perform reasonably well.

### 9.1.2 Limitations

The project also faced some limitations. The dataset is relatively clean and well-curated, and it contains only individual leaf images, which makes the classification task easier. Consequently, models often achieve high accuracy after a single training run, and results may overestimate real-world performance, where leaves could be partially occluded, affected by multiple diseases, or embedded in natural plant and environmental contexts. Additionally, very large models like **VGG16** could not be trained due to excessive memory requirements, illustrating practical hardware constraints.

## 9.2 Lessons Learned

Firstly, our group gained experience building models from scratch, implementing custom architectures, and also leveraging pretrained models such as ResNet18 and MobileNet, understanding the benefits and trade-offs of each approach.

Secondly, our group also became familiar with the entire model training process, including the forward pass to compute predictions, backward pass and backpropagation to calculate gradients, and optimization using algorithms like Adam, as well as tuning hyperparameters such as learning rate and batch size to achieve optimal performance.

Finally, we learned how to evaluate models systematically, using metrics such as accuracy and F1-score to measure both overall correctness and the balance between precision and recall. Comparing different models helped us understand the trade-off between performance, model size, and computational efficiency, which is crucial when deploying models in resource-constrained environments.

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[3] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[4] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[5] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *arXiv preprint arXiv:1801.04381*, 2018.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.