Image-Based Plant Leaf Disease Recognition with InceptionV3 Network

Senior Honors Thesis

Presented in Partial Fulfillment of the Requirements for Graduation with Distinction in the College of Engineering of The Ohio State University

By
Zhengqi Dong
Department of Computer Science and Engineering

Advisor:
Professor Darren Drewry
Department of Food, Agricultural & Biological Engineering

The Ohio State University
2020.08.08

Thesis Committee
Darren Drewry, Advisor
Committee Member
Committee Member

Abstract

Most traditional plant disease diagnosis strategies depend on human visual observation and inspection. However, this approach is time-consuming and requires significant human effort and expert knowledge. The recent advances in computer vision and deep learning provide a potential pathway to developing a plant disease diagnosis system that allows rapid detection of disease across large spatial areas with minimal human intervention. In this study, we developed a deep learning approach for plant leaf disease classification problems and conducted a range of experiments to quantify the performance of several state-of-the-art neural network architectures, including ResNet50, InceptionV3, and NASNet. All of the experiments were trained on the PlantVillage dataset with 54305 images in total, spanning over 38 plant disease classes. We evaluated four different performance metrics to assess each architecture: accuracy, precision, recall, and area under the curve (AUC). Our results showed that the InceptionV3 neural network architecture outperformed all other Convolutional Neural Network (CNN) architectures (ResNet50, NASNet-Large, NASNet-Mobile, MobileNet-v3-small, and MobileNet-v3-large) and produced a training accuracy of 94.14% and 97.94% over 6 epochs and 40 epochs of training, respectively. These results suggest that CNN architectures broadly, and the InceptionV3 model specifically, is capable of remote and automated plant disease detection. These results point to exciting future applications in lightweight mobile phone applications or backend workstation developments for plant leaf disease recognition problems.

Acknowledgments

The author would like to thank Professor Dr. Darren Drewry for the helpful advice and mentoring, which provides great support for the successful accomplishment of this project and thesis.

Vita

2017 to present .....B.Eng. Computer Science and Engineering,
The Ohio State University

Fields of Study

Major Field: Computer Science and Engineering

Minor: Statistics

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Plant disease is a significant threat to food security, impacting both agricultural yield quantity and quality in an unpredictable way. The Food and Agriculture Organization estimates that diseases, insects, and weeds cause approximately 25% of global crop losses. In the United States, roughly $40 billion in crop yield losses are caused by plant diseases annually. In most advancing countries, smallholder farmers play an essential role in global food production, but the traditional approaches in plant disease detection are expensive and time-consuming, which makes it impractical for smallholder farmers to adopt [29][41]. Therefore, an accurate and affordable plant disease diagnosis system is critical for identifying the plant disease at an early stage, allowing for mitigation efforts to control disease propagation. The recent improvement in Deep Learning technologies and the large scale of collected datasets showed a promising approach to achieve this goal cost effectively and efficiently [1,31].

In this thesis, we demonstrated a clear pathway to conduct the deep learning approach in solving plant disease classification problems and the experimented methodologies during the study. The remainder of this thesis is organized in the following manner. Chapter 1 will give a brief overview of plant disease history and the motivation that drove this research. Chapter 2 will discuss some of the achievements that had been made in the past and some experiences that can be valuable to the study. Chapter 3 will discuss some fundamental concepts of deep learning technologies and the state-of-the-art deep neural network (DNN) architecture adopted in the study. Chapter 4 will illustrate the result of this study and some insightful discoveries that can be contributed to other researchers. Chapter 5 summarizes the essential findings and issues that need to be addressed in the future.

Background and Motivation:

Plant diseases are caused by various plant pathogens, including viruses, bacteria, oomycetes, fungi, nematodes, and parasitic plants. Plant diseases can be an issue for any plant system, but those affecting agricultural systems can have a particularly detrimental impact on human livelihoods and health. For example, the late blight of potatoes, a disease caused by Phytophthora infestans -- a fungus-like oomycete pathogen, was first discovered in the early 1840s in Ireland. After the epidemic outbreak, about 1 million people died from starvation, and about 2 million

people immigrated to other countries to escape starvation. This example points to the tremendous human impacts that major agricultural disease outbreaks can cause. More common are less massive outbreaks that result in loss of yield, detrimental effects on economies, and particularly devasting effects on small farm holders or subsistence farmers [27].

Most traditional strategies in plant disease monitoring and tracking depend on visual inspection. In some cases, disease detection is aided by microscopic observation at the molecular level [29]. These approaches tend to be accurate but are limited in the spatial extent to which they can be applied and can be biased by the previous experiences of the person making the visual inspections [29]. These methods are likewise costly and not practical for the broad agricultural community. What is required is an accurate and affordable plant disease diagnosis system that will help farmers, especially smallholder farmers, to detect plant diseases at early stages, across extensive fields, and with the ability to be deployed many times throughout a growing season

A major consideration for agriculture in the coming decades is the need to provide an early warning and forecast for effective prevention and control of plant disease. A primary factor in this effort could be plant disease detection, which would prevent significant economic losses and enhance smallholder agriculture's resilience. The development of accurate and affordable plant disease diagnosis systems is, therefore, an urgent priority.

Motivated by the recent advances of machine learning and computer vision systems, numerous achievements have been made in deep learning since the success of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [11]. It has been demonstrated that the deep learning approach can achieve more accurate results than the traditional approach for some plant disease recognition problems [14,31]. These initial studies merit further research and extension to different diseases, cropping systems, and geographic contexts. As well, much less progress has been made in the more complicated problem of diagnosing disease severity level and in differentiating the type of disease that is occurring – critical for adequately managing or mitigating a disease outbreak situation.

Chapter 2: Related Works

In this section, we will discuss some of the achievements of the past studies, the dataset that applied to the study, and the motivation that led us to choose the deep learning approach over other methods or techniques.

Before our proposal, a plethora of work that has done in the field of plant disease image recognition. The most common techniques that had been investigated including support vector machine(SVM) [10,35], K-nearest neighbor(KNN) [18], and random forest(RF) [24]. However, most of those techniques were machine learning-based, which often requires an extensive amount of works for feature extraction. Leaf disease image recognition is a challenging and complex problem owing to the trivial distinction between disease symptoms and complex background conditions. Finding a good feature mapping function that can accurately identify the type of disease for a given image requires a lot of human effort and expert knowledge to overcome those challenges. Besides, if the type of problem has changed, the researchers would need to redesign the feature vector for the target object, and designing a good feature vector is not a trivial task. Thus, an automatic image recognition approach is necessary.

These deep learning approaches shed the burden of feature engineering with the powerful capability to automatically learn the features from data. Within recent decades, granted the availability of a large number of datasets, high-performance GPUs resources, and novel algorithmic breakthroughs, deep learning thrived in the field of image recognition and object detection [33]. The deep learning approach had been quickly propagated and is becoming a popular tool in plant disease diagnosis. In 2015, [22] used a 3-layer CNN learning system that trained on a total of 800 cucumber leaf images, and they achieved an average of 94.9% accuracy under 40 epochs of training. In 2016, [31] conducted an empirical study on the AlexNet and GoogLeNet DNN architecture and achieved a 99.35% classification accuracy on the PlantVillage dataset over 50k sample images over 30 epochs of training. In 2019, the author of [15]hand-designed a 9-layers deep convolutional neural network(CNN) and trained on the PlantVilllage dataset with six different types of augmentation methods, and the result of the proposed model achieved 96.46% classification accuracy over 3000 epochs of training.

All of those outstanding results showed a clear path to the approach of deep learning-based image recognition, which outperformed the state-of-the-art machine learning approaches on a large scale [15]. However, with ten thousand disease images, the early proposed approach would usually take over 40 or even thousands of epochs to reach the state-of-art result, and that likely could take over 20 hours or even a week. In our study, we compared the performance of three different state-of-the-art deep neural networks (DNN) models (ResNet50, InceptionV3, and NASNetMobile) to increase the learning ability along with decreasing the span of training time. The study of our result shows that, with the pre-trained InceptionV3 model, it is possible to shorten the training time under 2 hours and maintaining comparable performance.

PlantVillege Dataset

The dataset used in this study was provided by PlantVillage [21], which contains 54,305 leaf image samples with 38 classes spread over 14 crop species and 26 diseases. Each image was cropped in a standard size of 256x256 in the RGB channel, and a sample of images for each category has shown in Figure 1. The label of those images was annotated by the plant pathology expert, and their labeled name had shown in Figure 2.

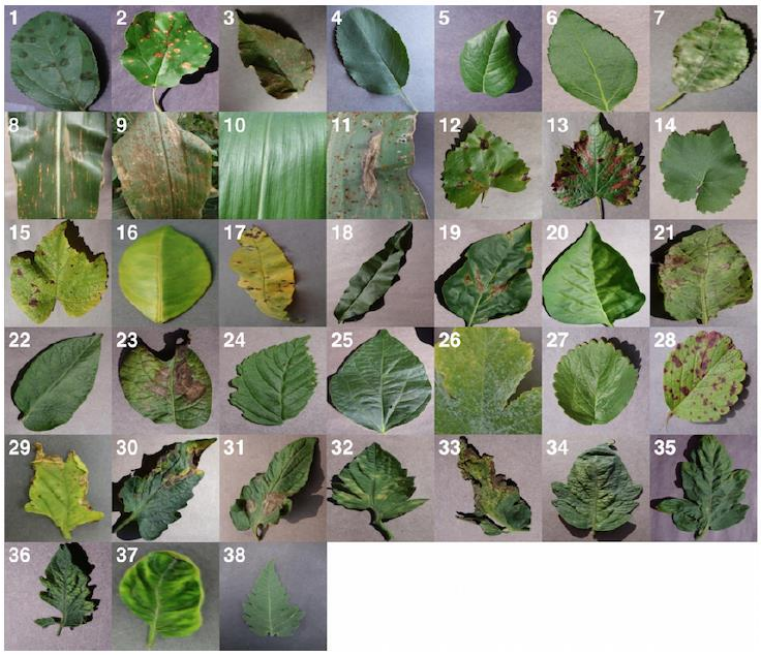

Figure 1: Leaf image samples for PlantVillage dataset [31].

```
Apple___Apple_scab                                  Pepper,_bell___healthy
Apple___Black_rot                                   Potato___Early_blight
Apple___Cedar_apple_rust                            Potato___Late_blight
Apple___healthy                                     Potato___healthy
Blueberry___healthy                                 Raspberry___healthy
Cherry_(including_sour)___Powdery_mildew            Soybean___healthy
Cherry_(including_sour)___healthy                   Squash___Powdery_mildew
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot  Strawberry___Leaf_scorch
Corn_(maize)___Common_rust_                         Strawberry___healthy
Corn_(maize)___Northern_Leaf_Blight                 Tomato___Bacterial_spot
Corn_(maize)___healthy                              Tomato___Early_blight
Grape___Black_rot                                   Tomato___Late_blight
Grape___Esca_(Black_Measles)                        Tomato___Leaf_Mold
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)          Tomato___Septoria_leaf_spot
Grape___healthy                                     Tomato___Spider_mites Two-spotted_spider_mite
Orange___Haunglongbing_(Citrus_greening)            Tomato___Target_Spot
Peach___Bacterial_spot                              Tomato___Tomato_Yellow_Leaf_Curl_Virus
Peach___healthy                                     Tomato___Tomato_mosaic_virus
Pepper,_bell___Bacterial_spot                       Tomato___healthy
```

Figure 2: Category path for the PlantVillage dataset

Chapter3: Methodologies

Comparison between Machine Learning and Deep Learning

In this section, we will provide some critical insight that explains why we believe the deep learning approach is the best approach for our study and why the model does not easily overfit the data even with a considerable scale model size.

The past traditional machine learning approach was mostly statistical-based modeling and required researchers to invent some intriguing algorithm to extract the feature. However, in the deep learning domain, the neural network does not require a hand-design of the feature extraction algorithm, and data itself powers the algorithm.

Initially, it might seem bizarre that a deep neural network (DNN) model can have millions or even billions of numbers of parameters and does not overfit the data. However, there indeed exist some differences between traditional machine learning and deep learning. As described by Dr.Yann LeCun, one of the pioneers in deep learning, in the 2018 IPAM conference, “Deep learning breaks some basic rules of statistics” [17]. In the past, whenever we say a perfect training accuracy in machine learning model (close to zero training error), we often will jump to the conclusion that the model is overfitting and try to stop the training. The traditional belief for machine learning is to find the “sweet spot” in Figure 3 for the model between the underfitting and overfitting in the U-shape curve, and the interpolation, exactly fitting the model to the training data, should be avoided for having poor performance or generalization abilities on the unseen testing data. However, this classical view does not hold the water in modern deep neural networking training. In practice, we saw many examples that a deep learning model can still achieve a good generalization ability even when the model is interpolating the training dataset [3,45].

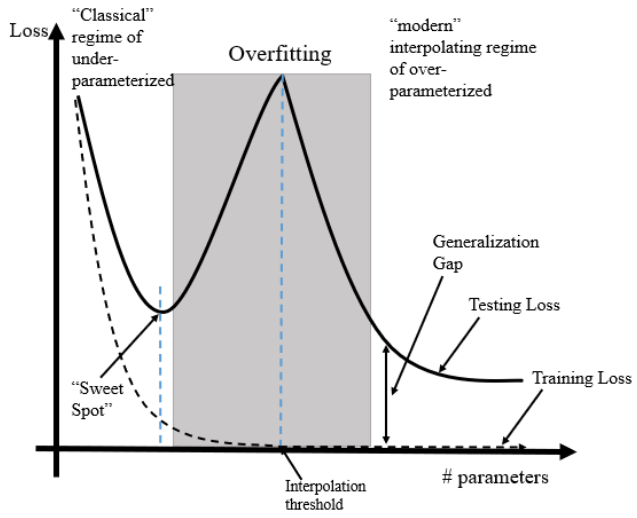

Figure 3: "Double-Descent" comparison between machine learning and deep learning, modified from [2]

The reason for this discordance between the classical machine learning domain and modern deep learning domain is because the generalization gap in traditional machine learning was based on the uniform laws of large numbers with capacity control, which can be defined as the following equation [4]:

$$\left| \mathbb{E}\left(L(f^*_{ERM}(\boldsymbol{x}), y)\right) - L_{emp}(f^*_{ERM}(\boldsymbol{x_i}), y_i) \right| < 0^*(\sqrt{\frac{c}{n}})$$

, where $f^*_{ERM}(\cdot)$ is the minimizer found by empirical risk minimization (ERM), which is the optimal function of a class of function space $\mathbb{H}$ that minimizes the risk over training; c is the measure of capacity, and n is the number of samples. What is on the left of the equation shows the generalization gap between expected loss and empirical loss, and the right-hand side of the equation defines the bound of the generalization gap.

This margin bound works well in the classical machine learning region. However, it does not apply to the models that reached the interpolation threshold [4], and the existence of such capacity-based generalization bound is still a debate [3]. The "double-descent" phenomenon

introduced by Mikhail Belkin gave some understanding for analyzing the over-parameterized model. It illustrated that deep learning models can achieve a good generalization ability even the model interpolated during training[2]. As you can see in the figure below, there is two way to avoid overfitting. On the left-hand side of the traditional or statistical machine learning domain, we try to prevent overfitting by decreasing the number of parameters (e.g., by introducing regularization terms that penalize the model's complexity). On the right-hand side of the modern interpolating domain, we avoid overfitting by increasing the data size or the number of parameters, often exceeding the number of training data, to over-parameterize the model. As you can see in Figure 3, the training loss continuously decreases as the model becomes more complicated.

Prevention of overfitting

Overfitting means the model fits well at training data (even outliers) but generalizes poorly in unseen data. For example, we used a 3-dimensional function to fit a small dataset generated by a 1-dimension function. In deep learning practice, a model with too much capacity, learning more function mapping from input to output, can learn too well and overfit the training dataset, which often occurs when the function makes the prediction that fits too close to the data point.

Dropout

Dropout provides an inexpensive approximation to training and evaluation for a complicated neural network and makes the model less sensitive to neurons' specific weights. The idea of dropout is motivated by human genetic reproduction, where the human offspring takes half of the gene from the mother and another half from the father, and some random mutation. This mixed set of genes intuitively can be optimized from the individual property of each side of the parent and to have more superior organisms [38].

The dropout operation is simple. Each time you add a dropout operation after a neuron network layer, a certain amount of random neurons will be dropped or discarded. The probability that a neuron will be removed depends on the preset dropout rate p, e.g., if p equals 0.5, then each neuron will have 0.5 of probability being dropped before feeding to the next layer. The

higher the dropout rate you use, the more neurons being dropped, and the stronger the regularization effect. During the training, the contribution to the activation of downstream neurons is temporally removed from the feed-forward path, and their weight updates are not applied to the neurons on the backward path. Also, the greater number of neurons being removed means you will have fewer training samples for the next layer, which can lead to an underfitting problem. Practically speaking, the typical range of dropout rate p should be between 0.2 to 0.5 for the hidden layer and 0.2 for the input layer [38].

Pre-training Model for Transfer Learning

Sometimes, we might not have access to a very large dataset on a specific domain or some high-performance computing resources. With transfer learning, we do not have to apply the machine learning technology from scratch with randomly initialized parameters but can use the pre-train model from a related domains as at "jump start". The idea of transfer learning is to retrain the last few layers of a pre-trained model (or source model), and the parameters of those layers will be randomly initialized and trained on a new dataset of interest that the model has not seen before (or target model). The pretrained model can be a model that had been trained by other researchers or scientists on a very large dataset, such as millions of images, for many days or weeks. This is a very effective and practical approach to reach a high accuracy in a short amount of time by building the experience upon others scussess. As the feature at the top level of the DNN model to be extracted from the low-dimensional representation that can be shared across related tasks, those common features can be directly used as the model's weight parameter to reduce the training cost, as well as the model errors. Whereas the layers in the lower level of the network tend to be extracted for the more a specific feature [32,43]. In our study, we added extra 1024 neurons of a dense layer to fine-tune the PlantVillege leaves disease dataset before transmitting it to the 38 neurons in the outputs layer, so the model can learn the new dimensional representation that is more suitable for our dataset. Notice that the effectiveness of transfer learning can be declined if the training target is less similar to the pretrained model. However, recent research also discovered that transfer learning could improve the generalization ability,

and even the transferred parameters from distant tasks can provide better performance than the randomly initialized parameters [43].

Data Augmentation

The idea of data augmentation is to make a series of changes to the original training data and to create more fake data, such that there is a higher likelihood of more possible features being encountered during the training process. The image augmentation methods are commonly broken into two categories: 1) photometric distortion, such as adjust the hue, saturation, contrast, and brightness, and 2) geometric distortions, such as image rotating, shifting, scaling, shearing, zoom in or out, horizontal or vertical flipping [6]. These operations are very effective to improve the generalization performance when we only have limited access to a large volume of training data. In certain sense, when we applied those operations to the training data it adds some noise to the images, so the dataset has more variability, and the model will have more opportunity to see various examples and more like to perform better in the future unseen data. In this study, a total of seven different augmentation methods were applied to the original training dataset, and the specific parameters for each method were listed in Table 1.

Model\Augmentation method	Rotation range	Width shift range	Height shift range	Shear range	Zoom range	Fill mode	Horizontal flip
ResNet-50	25	0.1	0.1	0.2	0.2	Nearest	True
InceptionV3	25	0.1	0.1	0.2	0.2	Nearest	True
NASNET	25	0.1	0.1	0.2	0.2	Nearest	True

Table 1: Image augmentation methods applied to the training dataset.

Optimization

A loss function is used to measure how bad or how good of a predicted result, and then the given data will deliver to the optimizer function. The goal of an optimizer function is to find a new set of parameters that can generate an improved guess that is expected to have a better result than before. As the guess getting better and better, the loss will converge and eventually

stop when evaluated metrics (e.g., accuracy, loss, validation accuracy) is lower than the predefined minimum tolerance.

Gradient Descent:

Gradient descent is the most common optimization algorithm used to minimize the cost function iteratively, and sometimes it is also known as vanilla gradient descent, or batch gradient descent. The goal for gradient descent is to find the best set of parameters θ that minimize the cost function [16] [34]. The algorithm for Gradient Descent is defined below:

1. Initialize weights randomly $\sim N(0,\ \sigma^2)$
2. Loop until convergence:
3. for $i \leftarrow 1\ to\ m$ do:
4. Compute Gradient, $\nabla J(\theta) = \nabla J_i(\theta) + \nabla J(\theta)$
5. $\nabla J(\theta) = \frac{1}{\mathrm{m}} \nabla J(\theta)$
6. Update weight, $\theta \leftarrow \theta\ -\ \eta \nabla J(\theta)$
7. Return weights

where m is the number of samples in the training dataset, $J(\theta)$ is the cost function, θ is the weight parameters to update, and η is the learning rate. The goal of gradient descent is

~~1. Initialize weights randomly $\sim N(0, \sigma^2)$~~

~~2. Loop until convergence ($\eta \nabla E(n) > c$):~~

~~2. Compute Gradient, $\nabla J(\theta) = -e^{(n)} x^{(n)}$~~

~~2. Update weight, $W(n+1) \leftarrow W(n) - \eta \frac{\partial J(\theta)}{\partial \theta}$~~

~~2. Return weights~~

~~, where η is the learning rate, c is the minimum tolerance, n is the number of iterations, E~~

~~The process for calculating gradient of cost function is show below:~~

$$J(\theta) = \sum_{i=1}^{m} l(h_\theta(x^{(i)}), y^{(i)})$$

$$\nabla J(\theta) = [\frac{\partial J(\theta)}{\partial \theta_1}, \frac{\partial J(\theta)}{\partial \theta_2}, \ldots, \frac{\partial J(\theta)}{\partial \theta_m}]$$

Formatted: Indent: Left: 0.25", No bullets or numbering

is the learning rate. The goal of gradient descent is to find a set of parameters that minimize the cost function as low as possible. The way to do this is to follow the opposite direction of the gradient, and that is why we use a negative sign in the step of updating weights.

$$\nabla J(\theta) = \frac{1}{2}[y' - y]^2 = \frac{1}{2}[\theta^T X - y]^2$$

Note: here we use mean square error as Cost function, but other functions are also work, e.g. squared loss, $l(y', y) = (y' - y)^2$, and absolute loss, $l(y', y) = |y' - y|$, for regression; 0-1 loss, $l(y', y) = \mathbb{1}_{\{y' \neq y\}}$,

is the learning rate. The goal of gradient descent is to find a set of parameters that minimize the cost function as low as possible. The way to do this is to follow the opposite direction of the gradient, and that is why we use a negative sign in the step of updating weights.

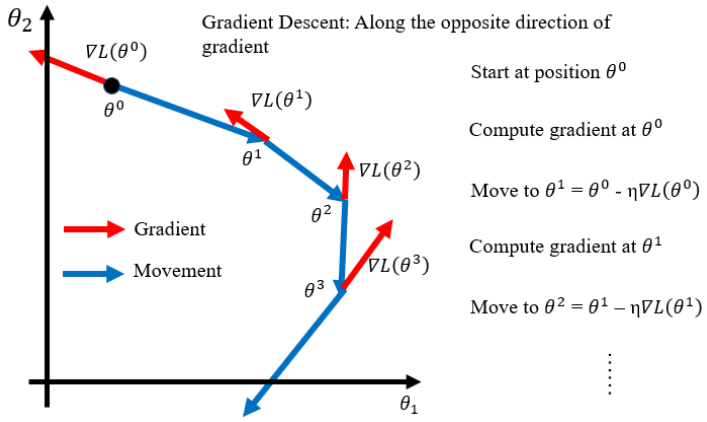

Figure 4: The leading direction of gradient descent algorithm

Commented [ZD1]: Delete the example, Simplify the document,

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is a special form of Gradient descent. Instead of updating the parameter vector θ after iterating throughout the whole training dataset, SGD updates the parameters for one sample at a time.

The Algorithm for SGD is listed as follow:

1. Initialize weights randomly $\sim N(0,\ \sigma^2)$
2. Loop until convergence:
3. for $i \leftarrow 1\ to\ m$ do:
4. Compute Gradient, $\nabla J_i(\theta)$
5. Update weight, $\theta \leftarrow \theta\ -\ \eta \nabla J_i(\theta)$
6. Return weights

Mini-batch Gradient Descent

The Gradient Descent algorithm takes the whole batch dataset to compute a single gradient step and update the parameter vector. This method is more accurate, as it has the immediate vision to all available training data, but it can be very computationally expensive as the training size grows to a million or billion examples and extremely underutilize the availability of parallel computation provided by the modern multicore architecture. The SGD is fast and allows the model to learn "online" – using a single example at a time, but it can be sensitive to the outlier. If the learning rate is not small enough to maintain the stability, the model can be oscillating around the local minimum and never converge.

Therefore, instead of taking a whole dataset or single data point to compute the gradient, the mini-batch gradient descent approach is used more often as a smaller sample of data (commonly use power of 2 batch size range from 32 to 256) can shorten the training time and ensures the model can still converge. The mini-batch gradient descent algorithm takes a batch of m' sample $\mathbb{B} = \{x^{(1)}, \dots, x^{(m')}\}$ to compute the gradient and update the parameter vector. Here is the algorism for computing the mini-batch gradient descent [17]:

1. Initialize weights randomly $\sim N(0,\ \sigma^2)$
2. Loop until convergence:

3. for $i \leftarrow 1\ to\ \left\lceil \frac{data\_size}{m'} \right\rceil$ do:
4. for $j \leftarrow 1\ to\ m'$ do:
5. Compute Gradient, $\nabla J(\theta) = \nabla J_j(\theta) + \nabla J(\theta)$
6. $\nabla J(\theta) = \frac{1}{\mathrm{m}'} \nabla J(\theta)$
7. Update weight, $\theta \leftarrow \theta - \eta \nabla J(\theta)$
8. Return weights

The batch size for mini-batch gradient descent is typically ranged from one to a few hundred and usually used as the power of 2 (e.g., 16, 32, 64, 128). The specific value for batch size depends on how much parallelization and memory your CPU/GPU can provide, and there is more advices about the strategy to optimize the training in Chapter 8.1.3 of [17]. Besides, many researchers' experiments show that a larger batch size (>128) can degrade the model's generalization ability [23,30]. With my experiences, the best approach to search for the optimal batch size is to start with a batch size of 1 and maintain a small learning rate (e.g., 0.0001), and then to tune the batch size graduately(e.g., 8, 16, 32), until reach to a point where the training time is desirable and the model still remains a good generalization ability.

Convolutional Neural Network (CNN)

Before the invention of CNN, object identification was a complicated task in computer vision. It is difficult because that an image is constituted by a series of pixels, which ranges from 0 to 255, and there is no single formula that could transform this series of pixels in the image to a "yes" or "no" answer.

Convolutional Layer

The motivation for the convolutional layer is to get a larger receptive filter such that the model can detect broader features from input images. A simple approach to process an image is to look over all the pixels in the images. However, that could slow down the training process extraordinarily. The image pixels are very sparse, and they can have many zero pixels that do not contain the important feature of the object that we want. The convolution operation involves using a filter and sliding that filter over the image to narrow down the underlying image to some

more specific and distinct features that are more unique to the object. The filter parameters learned from data make it easier for the detector to fine the important features (e.g., edges, shapes) that are useful to determine the output and possible to build a deeper neural network [44]. A convolution operation is commonly known as filtering, and the output from a convolution is called a feature map or activation map. The convolution operation is originally motivated by the cross-correlation operation, and an example of a 2D convolutional operation can be is defined in the equation below [17]:

$$S(i,j) = I(i,j) * K(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

where I denotes the input, K denotes the kernel or filter, S is called feature map, (i, j) indicates the entry position in a 2-D matrix, and m and n indicate the respective length and width for the kernel. For example, if we assume to apply convolution operation on a 2 by 2 kernel, the first entry of the feature map can be calculated as Figure 5.

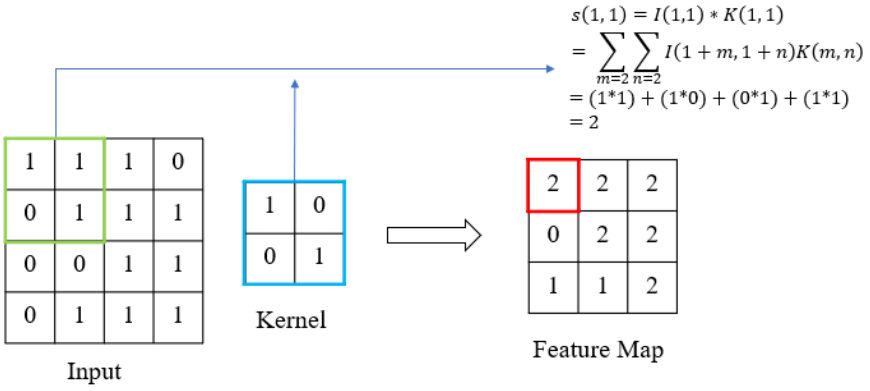

Figure 5: Convolutional Operation over 2x2 kernel

Pooling Layer

The pooling layer gives a way of compressing an image by replacing the output of the feature map with a summary statistic of nearby outputs. For example, in max-pooling, the max statistic will be used to compute the result of a sliding window, so the feature map is

downsampled in such a way to reduce redundant pixels and helps in faster training and lower memory consumption. In average pooling, the average statistic will be used to compute the result of a sliding window, so the feature map is averaged before delivered to the next layer. Therefore, by using the pooling layer in building a deep neural network, the information that an image carries can be greatly simplified [17].

Activation Functions

A deep learning neural network has no difference to the preliminary multi-layer perceptron with a linear activation function, which is trained based on a stack of linear operations. To classify those non-linear separable problems in low dimension, we need a nonlinear transformation to cast the training data to a higher dimensionality and make the problem to be classifiable. As described in Cover's theorem: "A complex classification problem is more likely to be linearly separable in high-dimensional nonlinear space than in low-dimensional input space" [8]. This section will introduce several commonly used activation functions in deep learning study, including softmax, sigmoid, hyperbolic tangent, ReLU (rectified linear unit) and its variances.

Softmax

The equation of SoftMax function is defined as below:

$$p_i = g(z_i) = Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

where z_i is the output for each linear combiner, and p_i is the probability produced for class i. More specifically, we can express the linear combiner as the function below:

$$z_j = \sum_{i=1}^{m} w_{ij} x_i + bias$$

where i indicates each neuron at the first layer, and j indicates each neuron at the second layer. The SoftMax function is frequently used in multiclass classification tasks, for which it has the desirable property of compressing the real number value into a range between 0 and 1 and ensures the result of all output probabilities will sum up to 1:

$$\sum_{i=1}^{m} p_i = 1$$

This befitting property can be visualized in an example of applying softmax in a multi-class neural network classification task, which showed in the figure below:

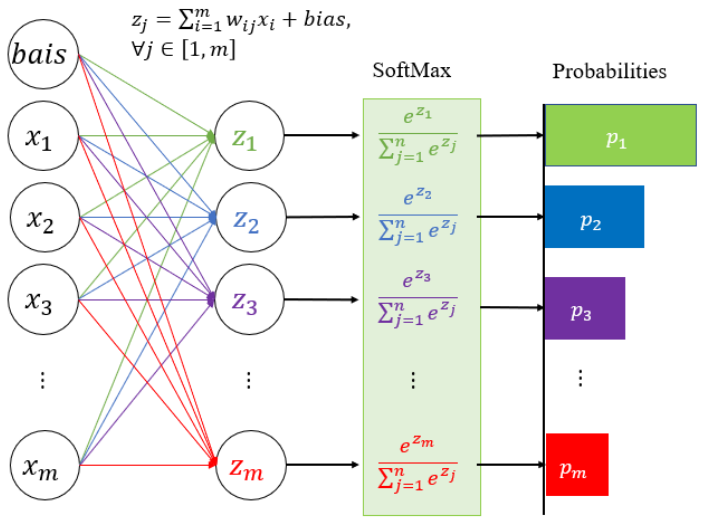

Figure 6: Illustration of a multi-class classification example with SoftMax

Sigmoid

The equation of sigmoid is defined as below:

$$g(z) = sigmoid(z) = \frac{1}{1 + e^{-z}} = \frac{e^{z}}{1 + e^{z}}$$

The first derivative of the sigmoid function is:

$$g'(z) = g(z)(1 - g(z))$$

With Sigmoid as an activation function, it compresses all the input values into a range between 0 and 1. With any input value greater than 0, the output result will be greater than 0.5; and any input value that is below zero, the output result will be less than 0.5; and any input value that is equal to zero, the output result will be 0.5. The sigmoid function is frequently used as the last

output layer in binary classification tasks. The graph for sigmoid and its derivative is showed as below:

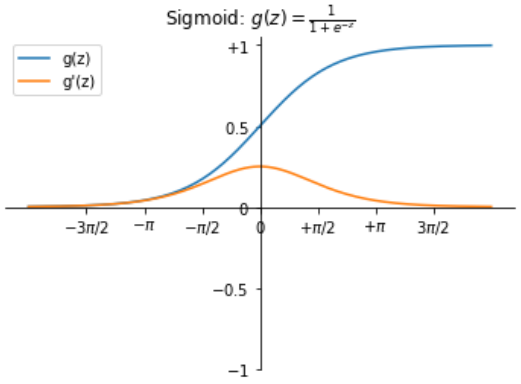

Figure 7: Sigmoid activation function and its derivative

Hyperbolic Tangent

The equation of hyperbolic tangent is defined as below:

$$\mathrm{g(z)} = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

The first derivative of the hyperbolic tangent is:

$$g'(z) = 1 - g(z)^2$$

With hyperbolic tangent as an activation function, it compresses all the input values into a range between -1 and 1. Any input value z that is greater than 0, the output result will be between 0 and 1; and any input value z that is less than 0, the output result will be between -1 and 0; and any input value z that equals 0, the output result will be 0. The hyperbolic tangent has similar property as sigmoid, but it is an origin symmetric function, and the output is zero-centered. The graph for hyperbolic tangent and its derivative is showed as below:

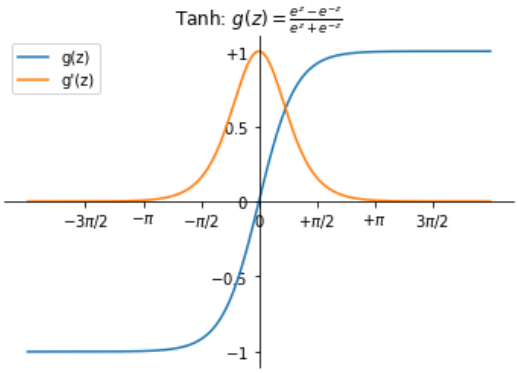

Figure 8: Tanh activation function and its derivative

Rectified Linear Unit (ReLU) and its variants

In contrast to the previous saturated activation function (e.g., sigmoid, tanh), many experiments show that non-saturated activation function – not resulting the gradient decay over layers, can solve the "exploding/vanishing gradient" problem and accelerate the convergence speed [42]. In this section, we will focus on the most popular non-saturated activation – Rectified Linear Unit (ReLU) and its variances.

ReLU is a less expensive operation compared to sigmoid and tanh, and it is frequently used to prevent the gradient vanishing problem in training a deep neural network. Also, some researchs' results show that a rectified neural network can provide a 2% absolute error rate deduction than the traditional sigmoidal unit [28]. With ReLU as activation function, the neural network will return any input value that is greater than zero without any change, and any input value that is less than or equal to zero will be discarded. The equation for ReLU is defined as below:

$$g(z) = Relu(z) = \max(0, z)$$

The first derivative of ReLU function is:

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & Otherwise \end{cases}$$

The graph for ReLU and its derivative is showed as below:

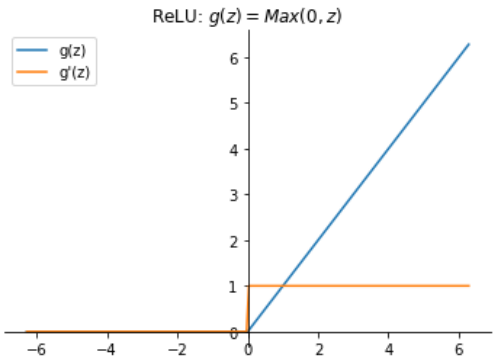

Figure 9: ReLu activation function and its derivative

There are several common variances to the ReLU are summarized below:

- Leaky ReLU (LReLU)

There is a "dying ReLU" issue about the ReLU activation function [26], where all the negative neurons will eventually become inactive as the network getting deeper. Because all the negative input values passed through ReLU will output zero, and the slope of ReLU in the negative range is also zero. It is similar to the vanishing gradient problem, where all the neurons with negative input values will always output zero, and the gradient will disappear eventually. Over time, those neurons can not learn anything meaningful. Therefore, instead of giving all the negative value to output zero, LReLU allow neurons to output negative value, and its equation is shown below:

$$g(z) = Relu(z) = \max(\alpha z, z), with\ \alpha \ll 1$$

where α controls the slope of function for $z < 0$. The larger the α will give a steeper slope. Typically, the hypermeter α will set to 0.01.

- Exponential Linear Units (ELUs)

ELU is similar to LReLU, but instead of giving a relatively flat slope for negative values, ELU uses a log curve, making its mean activation closer to zero and resulting a more robust model to the noisy data. Empirically speaking, ELU provides a faster training and better

performance in classification accuracies than using ReLUs and LReLUs on a neural network with more than five layers [7]. The equation for ELU is shown below:

$$g(z) = \begin{cases} z; & if\ x > 0 \\ \alpha(e^x - 1), with\ \alpha \ll 1; & Otherwise \end{cases}$$

- Gaussian Error Linear Unit (GELU)

GELU is a high-performance nonlinear activation function introduced this year in 2020. It was proved to have a better performance over the previous ReLU and ELU activation functions across many fields of deep learning tasks [20]. The equation for GELU is shown below:

$$g(z) = zP(Z \leq z) = z\Phi(z) = z \cdot \frac{1}{2}\left[1 + \text{erf}\left(\frac{z}{\sqrt{2}}\right)\right]$$

where $\Phi(z)$ is the standard Gaussian cumulative distribution function (CDF), and erf stands for Gaussian error functions. The overall comparison for the above four activation functions is shown below:

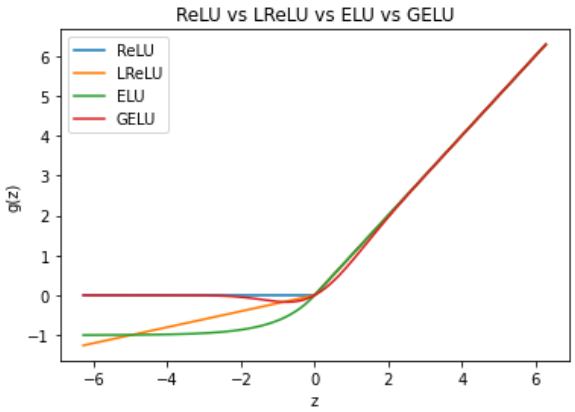

Figure 10: Comparison of four ReLU related activation functions

Classic Deep Learning Architecture

ResNet-50

Residual neural network(ResNet) was published in 2015 by [19]. Until now, it has been cited by over 55k times and had won 1st place on various image detection competitions, such as ILSVRC 2015 classification task, ImageNet detection, ImageNet localization, COCO image detection, and COCO segmentation. The ResNet architecture proposed a novel approach in building a deep neural network model by stacking a number of residual blocks without adding extra parameters nor computational complexity. A schematic diagram of ResNet-34 is shown in Figure 7, where the solid arrows denote the skip connection, and the dotted arrow denotes a special skip connection for bottleneck building block across over different dimension input and output layer. Such a genius approach greatly helps the prevention of vanishing/exploding gradients and the degradation problem where the generalization ability can get saturated as the network depth increases.

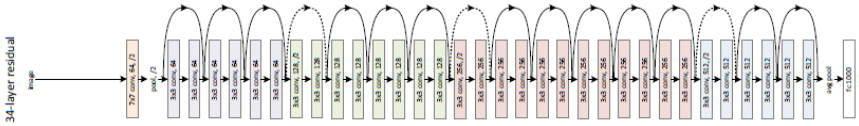

Figure 11: Schematic diagram of ResNet-34 architecture [19]

InceptionV3

InceptionV3 [40] model is the third edition of the Inception micro-architecture that was first introduced by Szegedy and other Googlers in their 2014 paper, Going Deeper with Convolutions [39], which is also known as GoogLeNet. The goal of the Inception module is to improve the utilization efficiency of the model's parameters and reduce the computational cost by factorizing large filter sizes into smaller convolutions and aggressive regularization via label smoothing. For example, in Figure 8, a 5x5 convolution filter is 25/9=2.78 times computationally expensive than a layer of 3x3 convolution, so a two-layer of 3x3 filters (with 3*3+3*3=18 parameters) can reduce 28% number of parameters than using a layer of 5x5 filter (with 5*5=25 parameters).

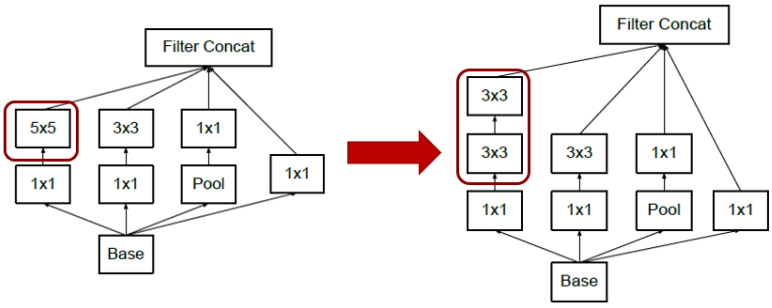

Figure 12: Replacing one 5x5 convolution with two 3x3 convolutions

The whole Inception-v3 model is comprised of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concerts, dropouts, and fully connected layers, shown in Figure 9. The result of this novel approaches outperformed the state-of-the-art performance on the ILSVRC 2012 classification challenge with 21.2% top-1 and 5.6% top-5 error rate while maintaining 2.5 times less computational cost.

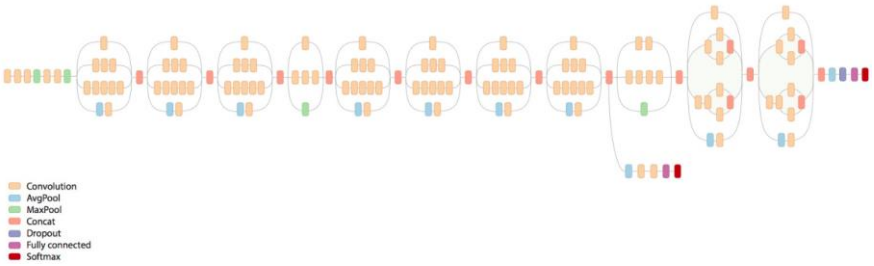

Figure 13: Schematic diagram of Inception-v3 [37]

NASNet

Neural architecture search neural network (NASNet) architecture was introduced by Google Brain in 2018 [48]. This network is different than the previous neural network which required an extensive amount of human effort in architecture engineering. Instead of designing the whole neural network architecture by a human being, reinforcement learning methods were

proposed to learn the best building blocks (or cells) from a related and smaller dataset (e.g., CIFAR-10) and then transfer the learned architecture to the larger dataset (e.g., ImageNet). The result of this training yields three candidate convolutional cells, NASNet-A (in Figure 10), NASNet-B (in Figure 11), and NASNet-C (in Figure 12) normal and reduction cells. In addition to the network architecture, a new regularization method called ScheduleDropPath was proposed and showed a huge improvement in the generalization ability. The result of this neural network architecture achieved state-of-the-art accuracy of 82.7% top-1 and 96.2% top-5 on the ImageNet dataset, which is 1.2% higher in top-1 than the best human ever invented while having 9 billion fewer FLOPS. The training efficiency of NASNet, with the use of 500 GPUs for 4 days resulting in 2000 GPU-hours on Nvidia P100, is approximately 7x faster than the original work in [47], with the use of 800 GPUs for 28 days resulting in 22400 GPU-hours on Nvidia K40.

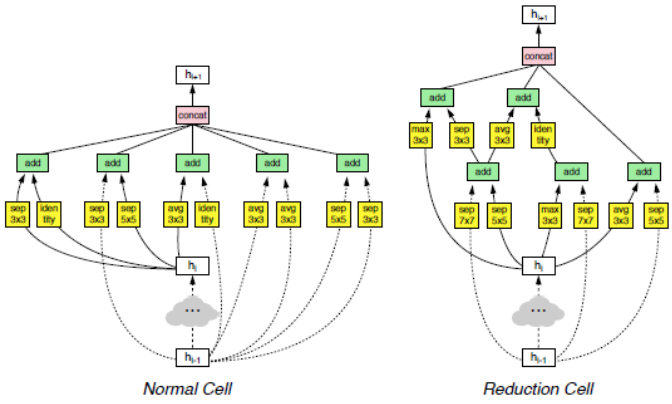

Figure 14: A diagram of the best convolutional cell (NASNet-A) that was learned from CIFAR-10 [48].

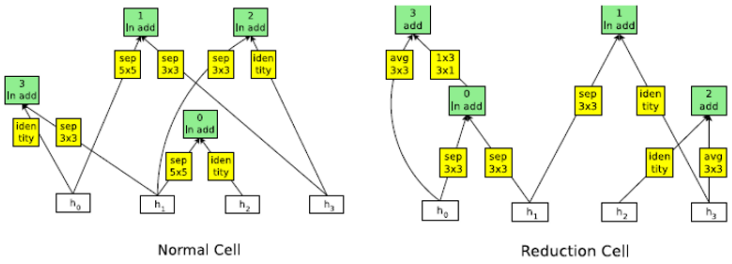

Figure 15: Architecture of NASNet-B convolutional cell [48]

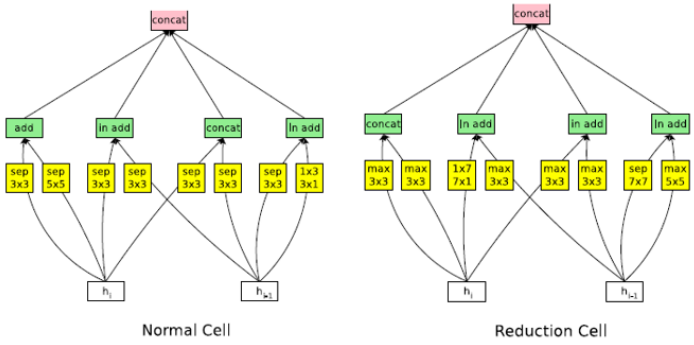

Figure 16: Architecture of NASNet-C convolutional cell [48].

Chapter4: Experiments and Analysis

Experimental Setup

To complete the study, all the experiments presented below were conducted on Ohio Supercomputer Center(OSC) Pitzer clusters and implemented in Python with CUDA-v11.0.3 as backend. Other specific hardware and software configurations had shown in Table 2.

Hardware/Software	Configurations
Memory	32GB
Disk	1TB
Computing Processing Unit(CPUs)	Dual Intel Xeon Platinum 8268s Cascade Lakes@2.9GHz x 48 cores
Graphics Processing Unit(GPUs)	Dual NVIDIA Volta V100
Operating System	Red Hat Enterprise Linux Server release 7.7
Python	3.6.5
TensorFlow	2.4.0
CUDA	11.0.3

Table 2: OSC Pitzer configuration with GPUs support (https://www.osc.edu/resources/technical_support/supercomputers/pitzer)

Training

In this study, the original PlantVillage dataset used in this study was divided into training and validation datasets by 80% and 20%, respectively, as it has shown the best performance in Table 3. The validation data is used for evaluating the performance after each epoch and did not involve in the training process. Before the training, each pixel of images was firstly normalized dividing by 255, and the input size for all models was set to 256x256 by default, and the batch size was set to 32, which is the highest accuracy performance based on the experimental result shown in Table 5. To speed up the training process, all CNN models in this study (including

ResNet50, InceptionV3, NASNet, and MobileNet) were used with the transfer learning approach, and all pre-trained models were previously trained on the ImageNet dataset [36]. During the transfer learning, a~~A~~ll layers in the pre-trained model were frozen to shorten the training time and reduce the computation cost (Because the network does not need to extract the generic features from scratch, and we believe this will not pose huge damage to the performance in terms of accuracy since the feature space of our dataset are comparable to the ImageNet). The last fully connected layer of all pre-trained models was taken for the fined-tuning purpose, and a dense layer with 1024 neurons was appended before transmitting to the 38 neurons in the outputs layer. Softmax was used as the activation function in the last layer, and the categorical cross-entropy was used for computing the loss function, which showed no significant difference to sparse categorical cross-entropy demonstrated in Table 6. During the training, SGD was used as the optimization method, and the learning rate and momentum were set to 0.001 and 0.9, respectively.

Evaluation Metrics

Choosing appropriate evaluation metrics is as important as choosing the learning algorithm, especially when the dataset is unbalanced. In this section, I will explain some differences in the statistics that I collected and shed some light on why they can be important to the study. The statistics that were collected after each epoch of training include time per epoch, cross-entropy loss, accuracy, precision, recall, and Area Under the Curve(AUC).

Accuracy

Accuracy is the most widely used and the simplest indicator for evaluating the number of correct predictions that had been made over the ~~training~~ data, but we must ask ourselves several questions: Does accuracy always perform well in evaluating the performance of a model? and when might accuracy not be an appropriate metric to use? In short speaking, accuracy might not be a good indicator for an unbalanced dataset. For example, if we have an unbalanced dataset that contains 999 negative samples and 1 positive sample in the dataset, as a result, the accuracy for this model will be 99.9%. However, the result is not fairly evaluated for the positive categories whereas the dataset only contains one positive sample, and the result can be

misleading and can be costly if the positive result here represents a COVID carrier or terrorist. The definition for accuracy is defined below as the percentage of corrected predictions [13]:

$$Accuracy = \frac{\#\ of\ corrected\ prediction}{Total\ samples}$$

As in the previous example, accuracy cannot explain well for an unbalanced dataset, and other evaluation metrics should take into consideration, such as precision, recall, or F1 score.

Precision

Precision is defined as the fraction of true positives examples among the total retrieved positive instances(a true positive(TP)+ false positive(FP)). The range of precision is between 0 and 1, and can be calculated as below equation [13]:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{Total\ Predicted\ Positive}$$

As the formula suggested, the precision metric evaluates the number of true positive prediction within total predicted positive samples and only need to take into consideration if false positive (or Type I error rate) is significant to the study.

Recall

The recall is defined as the proposition of TP examples amount the total true positive(TP + false negative(FN)) and is also known as sensitivity or True Positive Rate(TPR). The range of recall is also between 0 and 1, and can be calculated as following equation [13]:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{Total\ Actual\ Positive}$$

As the formula suggested, the recall should be taken into consideration if the FN is significant to the study.

F1 Score

F1 score is a weighted harmonic mean of recall and precision and had been widely used for many machine learning algorithms. The equation is defined below [13]:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

AUC

The area under the curve(AUC) is a popular metric for evaluating how good our model performed in distinguishing between classes, and sometimes it is also known as (Area Under the Receiver Operating Characteristics)AUROC. Each recall and precision will have a corresponding instance of AUC, The AUC has values ranged between 0 and 1. The higher the AUC, the better the separation capacity of the model in distinguishing between classes. Ideally, if AUC is 1, the model can perfectly distinguish between positive and negative samples; on the other hand, if the AUC is 0, the model has no separability [12]

Overall, accuracy is commonly used for the balanced dataset. For an unbalanced dataset, if the false positive is more significant than the false negative, we should use recall as an indicator, otherwise, precision should be used. In the case that both false positives and false negatives are important to us, we should consider the F1 score for the evaluation purpose, and the AUC curve is a valuable tool to visualize the model's separability on a skewed dataset.

Intersection over Union (IoU)

IoU is an accuracy evaluation metric that is commonly used in image object detection tasks. Unlike the traditional classification or object recognition tasks, the accuracy was measured based on the number of predicted results matched to the ground-truth labels. It is hard to expect two bounding boxes will align exactly in the same coordination x-y axis. The IoU used the ratio between the predicted bounding box and ground-truth bounding box to measure the accuracy. Specifically, it measures the proportion of intersected area between two bounding boxes over the area of their union. The more overlapping between two bounding boxes indicates a better-predicted result. This can be expressed as the equation below [11]:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

, where B_p is the predicted bounding box for detection, and B_{gt} is the ground truth bounding box.

Train/Valid/Test Dataset

In deep learning development, we could not initially come up with the optimal configuration parameters. Therefore, applying deep learning is a very repetitive process where we must follow the development circle from generating the idea, training the model, evaluating the performance, and over and over again. Therefore, one thing that can expedite the speed of the development process is to improve the efficiency in going through that cycle and setting up the appropriate partition ratio of the dataset in terms of training, validation, and testing.

The development dataset is also known as a held-out validation set or development set, but for brevity, we commonly just call it the “dev set” or “valid set”. A valid set is what we use to compare different models' performance and see which one performs the best during the development stage. Once we have the final model that we want to evaluate, the test set will be used to produce an unbiased estimation of the unseen data.

In many applications of the machine learning algorithm, we often start with the 60/20/20 train-dev-test split, or 70/30 train-test split (if you do not need an explicit dev set). However, in the modern deep learning application, or the big data era, we might have a million examples in total, and taking up 20% of the dataset for testing can be excessive. Since the goal of the dev set is for comparing the performance of a different algorithm, sometimes even just 1% of the dev set and 1% of the test set can suffice that purpose. (Note: A lot of people are usually confused by the purpose between valid and test set, so I add more illustration here, but if you do know, just skip the rest of the paragraph. Say If you have two models you want to use for comparing linear model and polynomial model, and you want to decide the best value to use for the regularization term. In this case, you need a validation dataset to evaluate the performance for different values of that hyperparameter to produce a model with the lowest generalization errors. Notice, in this case, you measured the generalization error multiple times on the validation set and adapted the model and hyperparameter to produce the best model for that set. Therefore, you cannot use this ‘validation dataset’ again on the final evaluation because the dataset had been viewed by the model. Instead, you will need to split up a test dataset for the final model that can produce an unbiased estimation [16])

Division of Training and Validation Sets

The size of training data is an important factor in deciding the quality of generalization ability of a model. In this study, we conducted 18 sets of experiments on 3 different DNN architectures (ResNet50, InceptionV3, and NASNetMobile) and 6 various ratios of the train-validation split to show the accuracy after 3 epochs of training. The result in Table 3 and Figure 8 show that there exists a clear trend that, as the size of the training dataset increased, the training accuracy for the model tends to increase. However, we also noticed that the performance of InceptionV3 does get degraded as the train to valid set ratio exceeds 90/10. We, therefore, believed that the 80/20 train-valid set split ratio is a good balance point to have a reliable model performance. (The reason that we decided to evaluate the performance after 3 epochs instead of 1, 5, 10, or other numbers, is because the InceptionV3 model performed so well on the training, and all result of accuracy getting close to 100.00% after 5-7 epochs of training.)

Train-Validation split\NN model	ResNet-50	InceptionV3	NASNetMobile
Train 20%, Valid 80%	0.1255	0.9860	0.2110
Train 40%, Valid 60%	0.0986	0.9880	0.1747
Train 50%, Valid 50%	0.1551	0.9905	0.2279
Train 60%, Valid 40%	0.1637	0.9911	0.1026
Train 80%, Valid 20%	0.1014	0.9926	0.2438
Train 90%, Valid 10%	0.2738	0.8659	0.3794

Table 3: Accuracy score across various experiment configurations ~~at the end of~~after 3 epochs of training.

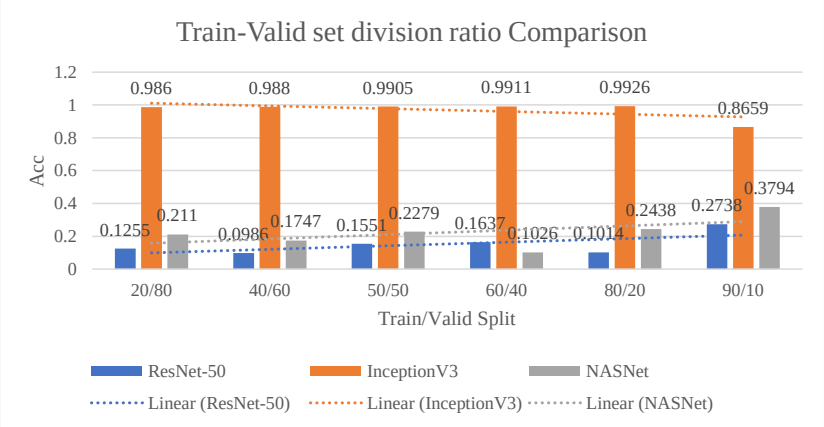

Figure 17: Comparing training ACC for different train-validation split ratio on 3 different DNN models after 3 epochs of training

Batch Size

The traditional SGD takes one example at a time and was trained sequentially. As the dataset grows and the availability of GPUs resource, the mini-batch SGD become more and more popular in training large scale dataset in a parallel and distributed fashion, where a subset of data can perform forward, and backward-propagation independently amount processors and synchronize the update through the global allreduce operation(a type of MPI communication) [9]. However, it is always challenging to decide the optimal best batch size to use in practice. If the mini-batch size is too small, we might face the problem of I/O communication overhead; on the other hand, if the mini-batch size is too large, the convergence rate can be reduced dramatically and can taking much longer for a single weight update. To determine the best batch size to use for this study, we conducted seven set of experiments for discovering the best mini-batch size to use. All the experiments were trained on InceptionV3 neural network with 0.2 validation split, and the result is listed in Table 6:

Batch size	Train Acc	Val Acc	Time/Epoch
32	0.8765	0.9544	1064
64	0.8594	0.9489	1159
128	0.8198	0.9289	1085
256	0.7509	0.9008	1164
512	0.6346	0.8554	1045
1024	0.4770	0.7300	1176
2048	0.3252	0.5662	1132

Table 4: Training accuracy on various batch size after first epoch

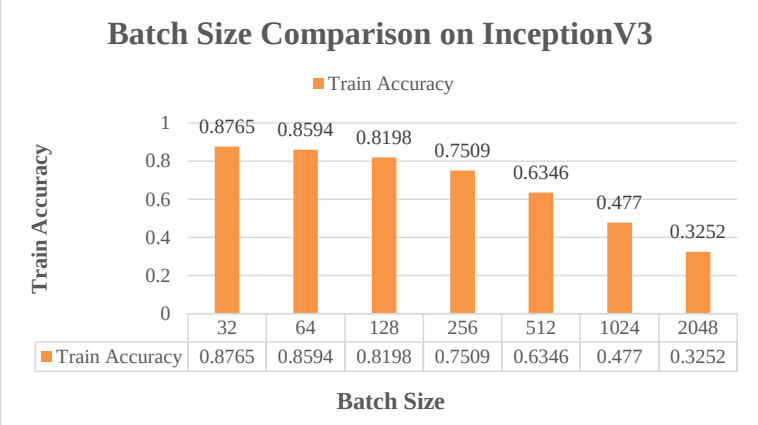

Figure 18: The performance evaluation of various batch sizes on InceptionV3 network after first epoch~~Training accuracy on various batch size~~

From the result of Figure 10, we observed that the training accuracy increased as we reduce the mini-batch size, but there is no significant impact on the training time per epoch. As suggested in the paper [25], the mini-batch size b should not be larger than the T/b iteration, where T represents the total number of steps for one epoch. We, therefore, believe 32 can be an appropriate balance-point for this study, and all the following experiments were performed with a batch size of 32.

Loss Function

Since there are 38 classes in the PlantVillage dataset, two common categorical loss functions were chosen for determining the best loss function in this study: Categorical Cross-Entropy(CCE) and Sparse Categorical Cross-Entropy(SCCE). CCE is a loss function that most used in multi-class classification, and the formula has defined below:

$$\text{generalized cross-entropy} = -\sum_{i}^{C} \boldsymbol{y}_i log\left(f_i(\boldsymbol{x_i}; \boldsymbol{\theta})\right)$$

where $\boldsymbol{y}_i$ and $f_i(\boldsymbol{x_i}; \boldsymbol{\theta})$ are the one-hot encoded label and DNN scores for each $class_i$ in C(as the activation function is always applied to the score function before computing the loss, the notation $f()$ is used to refer to that activation function). The CCE applied the one-hot encoding to compute the target score. For example, considering applying neuron network to train a model with 5 categories. The label can be represented as [0, 0, 1, 0, 0], and the predicted score can be [.2, .3, .5, .1, .1]. SCCE is similar to CCE except the integer encoding is used to replace the one-hot encoding. As the same example of 5 categories, the sample label can be represented as [2], and the predicted score can be [.5]

For comparing two loss functions, 6 sets of experiments were conducted, and the result in Table 7 shows that there are no significant differences between two-loss functions in our problem. For retaining all the information(CCE can measure both top-1 error and top-5 error) and considering the suggestion in [46] stated CCE is more robust in combating the noisy labels, CCE was chosen for the rest of the experiments.

Loss Function \ Evaluation metrics	Time/Epoch (sec)	Acc after 1st epoch	Acc after 25 epochs
Categorical Cross Entropy(CCE)	1426	0.4909	0.9965
Sparse Categorical Cross Entropy (SCCE)	1450	0.4881	0.9965

Table 5: Loss function comparison between CCE and SCCE

CNN Models Selection

Different CNN models have their own advantages on the different image recognition task. The CNN models that were chosen for this study were guided by the research result [5], which provides a comprehensive benchmark analysis of over 40 kinds of state-of-the-art CNN models that had been published before 2018. For our study, we selected some models that require more computing resource and more suitable for backend workstation, such as ResNet-50 (134.3M trainable params), Inception-V3 (75.5M trainable params), and NASNet-Large (264.3M trainable params); and some models that are less complicated and more efficient for mobile phone embedded applications, such as NASNet-Mobile(69.2M trainable params), MobileNet-v3-small(51.4M trainable params), and MobileNet-v3-large(83.9M trainable params).

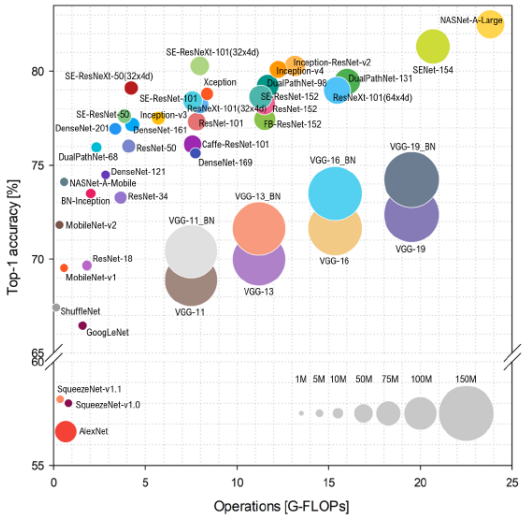

Figure 19: Ball chart for Top-1 accuracy vs. computational complexity (ball size represents the model complexity) [5]

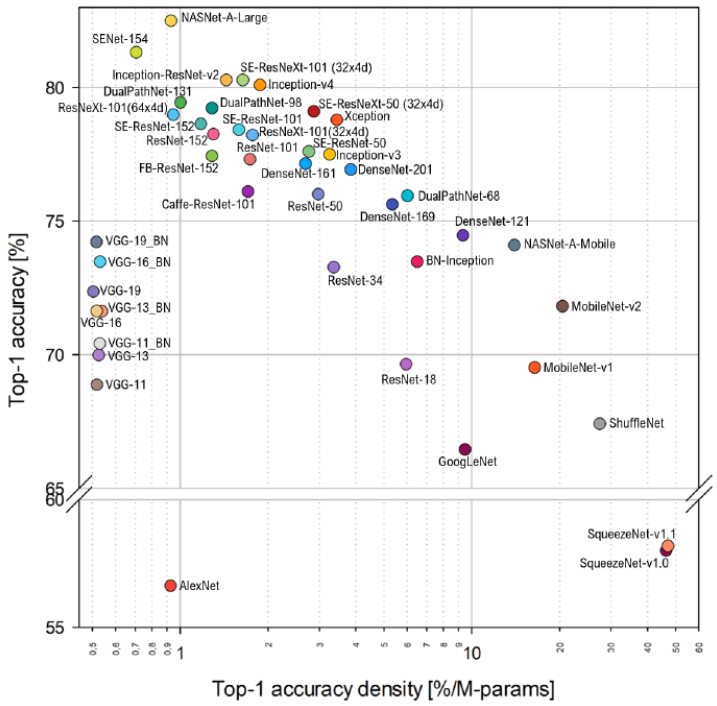

Figure 20: Top-1 accuracy vs. Top-1 accuracy density (measured by how efficiently each model uses its parameters) [5]

The backend workstation can provide more computing power for training models that have high computational complexity and requires days of training. From the benchmark result in Figure 15, we noticed the NASNet-Large model has the highest top-1 accuracy and computational complexity -- measured by the number of floating-point operations (FLOPs). Thus, we believe the NASNet-Large model can be a good option for our study. Besides, we noticed that some DNN architectures (e.g., NASNet-Mobile, and other MobileNet series) tend to have a comparatively low computational complexity (<5G-FLOPs) while maintaining a good top-1 accuracy (>70%). By looking at the top-1 accuracy vs. accuracy density plot in Figure 16, we can see that amount the most efficient DNN models -- have accuracy density greater than

10%/M-parameters, NASNet-Mobile and other MobileNet-like architectures tend to perform better in terms of the top-1 accuracy (>70%). Those properties are more desirable for the mobile embedded systems, so we decided to apply NASNet-Mobile, MobileNet-v3-small, and MobileNet-v3-large models to analyze the performance on the PlantVillage dataset. Also, several most popular CNN models (e.g., Inception-v3 and ResNet-50) that had shown extraordinaire performance in the ILSVRC ImageNet recognition competition were also tested during our CNN experiments. Those models do not have the highest accuracy shown in Figure 9 and Figure 10, but they were shown to be able to reduce the computational complexity below 5 G-FLOPs while maintaining a comparatively high accuracy (higher than 75%), so we believe Inception-v3 and ResNet-50 are good candidates for our study as well. Another factor that we had taken into consideration during the model selection is the availability of CNN models that were supported by the TensorFlow backend, without that support we will have to spend huge amount of time in designing and training the neural network model from scratch.

CNN Model Performance Evaluation and Analysis

The main objective of our study is to evaluate the performance of existing state-of-the-art CNN models and examine the success of the best deep learning architecture in the context of the plant leaf disease recognition problem. As mentioned at the beginning of Chapter 4, all the deep learning models used in this study were trained by applying the transfer learning approach.

All experiments in our study were carried out on the PlantVillage dataset. The evaluation metrics that had mentioned in Chapter 4: Evaluation Metrics were given in Table 6, including accuracy, loss, precision, recall, AUC on both training and validation datasets. As seen in Table 6, after 40 epochs of training, the InceptionV3 model achieved the best performance with 97.94% and 96.04% on the training and validation dataset, respectively, and this result outperformed other models by a large amount. Therefore, the InceptionV3 is a good candidate to be considered for heavily loaded backend application.

As seen in Table 7, we provide other information about the models (e.g., Input size, total parameter, trainable parameter, throughput, and training accuracy after 40 epochs), and it reveals the relationship between the model complexity, training speed, and accuracy. We see that the

largest CNN model in our study, NASNetLarge (with 350M paras), and the smallest CNN model, MobileNet-v3-small (with 52M params), have similar performance in terms of accuracy. Therefore, there is no obvious linear correlation between the model complexity and their generalization ability. Besides, we observed that the InceptionV3 model has the highest classification accuracy (97.84%) amount the three CNN models (ResNet-50, InceptionV3, and NASNetMobile) that were comparable in terms of throughput, which suggest that the InceptionV3 model could produce high performance in the mobile device application as well.

As seen in Figure 16 and Figure 17, there is no obvious sign of overfitting in any of the six models that we choose. Amount the six sets of comparison, the InceptionV3 model shows the best performance in terms of the training and validation accuracy. In Figure 16, we see that the InceptionV3 can achieve above 94% accuracy by less than 10 epochs of training, in contrast to other models that were struggling to reach the 60% accuracy even after 40 epochs of training, and the progression line for both accuracy and loss tends to fluctuate more violently. Thus, we believe that the InceptionV3 model offers the fastest learning efficiency in our study and, therefore, is the most suitable model to be used in plant leave disease recognition. Besides that, excluding the NASNetLarge model, the performance of all other models shows an increasing trending, in Figure 16 and Figure 17, even after 40 epochs, and that trending particularly obvious amount three models, InceptionV3, NASNetMobile, and MobileNet-v3-large. Therefore, we believe that these three models can have the potential to improve and achieve higher performance if we extended the training period above 40 epochs.

Model	Train accuracy %	Validation accuracy %	Training loss	Validation loss	Precision %	Recall %	AUC%	Valid precision %	Valid recall %	Valid AUC %
ResNet-50	17.68	15.90	2.9988	3.1075	48.16	2.05	79.51	0.00	0.00	77.80
InceptionV3	97.94	96.04	0.0587	0.1237	98.15	97.77	99.95	96.41	95.81	99.78
NASNetMobile	48.27	47.68	1.7768	1.8007	78.07	32.87	93.87	81.57	30.08	93.78
NASNetLarge	9.27	9.87	3.3834	3.3714	0.00	0.00	69.08	0.00	0.00	69.20
MobileNet-v3-small	17.36	17.23	2.9898	2.9602	55.00	0.72	79.90	24.13	0.06	80.64

MobileNet-v3-large	22.45	20.96	2.8342	2.8540	57.39	1.95	82.81	33.33	0.07	82.87

Table 6: DNN models' performance comparison after 40 epochs of training

DNN Model	Input size	Total Parameters	Trainable parameter	Throughput (s/epoch)	Train acc after 40 epochs(%)
ResNet-50	256x256x3	157,845,414	134,257,702	1099-1347	17.68
InceptionV3	256x256x3	97,340,230	75,537,446	1095-1374	97.94
NASNetMobille	256x256x3	73,515,706	69,245,990	1092-1270	48.27
NASNetLarge	256x256x3	349,197,944	264,281,126	1772-2135	9.27
MobileNet-v3-small	256x256x3	52,950,166	51,420,198	1894-3271	17.37
MobileNet-v3-large	256x256x3	88,152,486	83,926,054	2074-3196	22.45

Table 7: Classic DNN models' capacity comparison

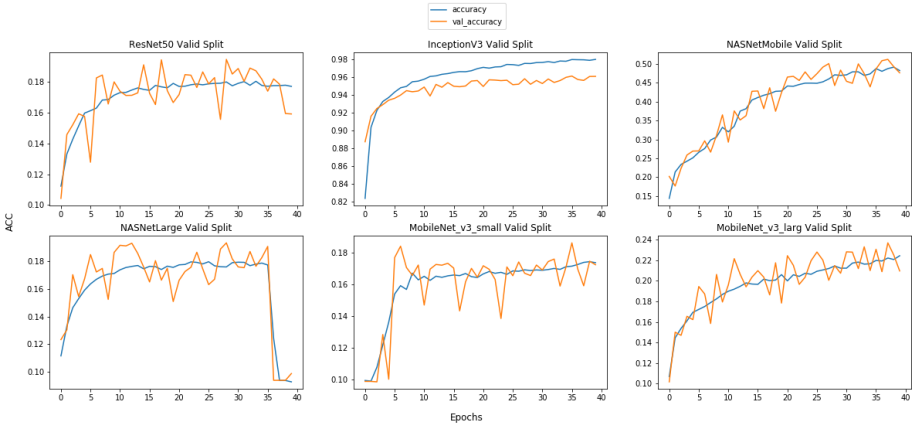

Figure 21: Models' accuracy and validation accuracy comparison

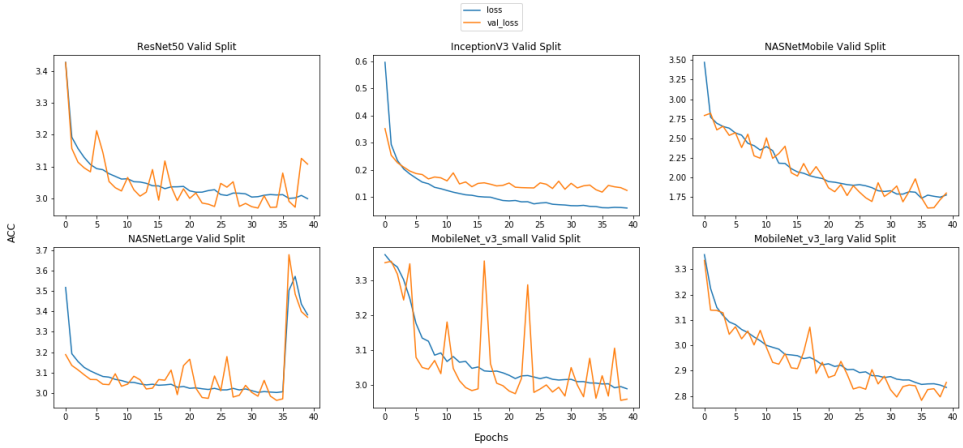

Figure 22: Models' loss and validation loss comparison

Chapter 5: Conclusion

In our study, the InceptionV3 deep learning architecture was tested to be the most suitable CNN model in applying to the image-based plant leaf disease recognition on the PlantVillage dataset. Throughout the study, we have conducted a comprehensive literature review on the past related research on the field of deep-learning-based plant disease detection and provided a series of empirical experiments in applying the techniques, including tuning the performance by varying train-valid set split ratio, pre-trained CNN models, loss functions, and batch size. All the experiments conducted in our study were trained on the OSC Pitzer cluster with 48 cores of Dual NVIDIA Volta V100 GPUs. The result of this study showed that the InceptionV3 neural network architecture outperformed all other candidates' CNN models. Specifically, we demonstrated in the experimental section that the InceptionV3 can achieve top-1 accuracy of 94.14% in training and 94.53% in validation over 2-hour training and 97.94% in training and 96.04% in validation over 16-hours, respectively, which suggest that the InceptionV3 model is suitable for both lightweight mobile applications and backend workstation purpose development within the context of plant leaf disease recognition.

Chapter 6: Future Works

Plant disease poses the main threat to the agricultural development of the world, and especially critical to the smallholder farmers in many developing countries. Therefore, affordable and accurate automatic plant disease detection can be valuable tools to provide early warning and forecast that mitigating the efforts to control disease propagation. The application of the deep learning approach has been growing quickly in the field of plant disease diagnosis. The result of deep learning approaches reported in many works of literature so far had shown a promising future. However, the deep learning approach is not omnipotent. There are some shortcomings and challenges that we have not mentioned in the thesis, such as its lack of adaptivity to the ever-changing environment, poor interpretability of the model, and the demand for a large volume of the dataset. Therefore, one important goal that our future work will be aligned on is the development of robust image detection that only requires a small set of image samples (<10). Alleviating the strict requirement of the size of the training image can make it more convenient to the smallholder farmers communities who want to extend our application to the new type of plant disease and speed up the automation pipeline deployment in an urgent condition.

In the end, we hope the result of this thesis can motivate and encourage more researchers to experiment with the technology of deep learning and apply it to the plant disease diagnosis problem that can bring the greatest benefit toward a more sustainable and intelligent farming and reliable food production.

References

[1] Ümit Atila, Murat Uçar, Kemal Akyol, and Emine Uçar. 2020. Plant leaf disease classification using EfficientNet deep learning model. *Ecol Inform* 61, (2020), 101182. DOI:https://doi.org/10.1016/j.ecoinf.2020.101182

[2] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *P Natl Acad Sci Usa* 116, 32 (2019), 15849–15854. DOI:https://doi.org/10.1073/pnas.1903070116

[3] Mikhail Belkin, Daniel Hsu, and Partha Mitra. 2018. Overfitting or perfect fitting? Risk bounds for classification and regression rules that interpolate. *Arxiv* (2018).

[4] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. 2018. To understand deep learning we need to understand kernel learning. *Arxiv* (2018).

[5] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. 2018. Benchmark Analysis of Representative Deep Neural Network Architectures. *Arxiv* (2018). DOI:https://doi.org/10.1109/access.2018.2877890

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *Arxiv* (2020).

[7] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ICLR* (2016). Retrieved from https://arxiv.org/abs/1511.07289

[8] Thomas M. Cover. 1965. Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers* EC-14, (1965), 326–334. DOI:https://doi.org/10.1109/pgec.1965.264137

[9] Nikoli Dryden, Naoya Maruyama, Tim Moon, Tom Benson, Marc Snir, and Brian Van Essen. 2019. Channel and Filter Parallelism for Large-Scale CNN Training. *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), 1–20. DOI:https://doi.org/10.1145/3295500.3356207

[10] M.A. Ebrahimi, M.H. Khoshtaghaza, S. Minaei, and B. Jamshidi. 2017. Vision-based pest detection based on SVM classification method. *Comput Electron Agr* 137, (2017), 52–58. DOI:https://doi.org/10.1016/j.compag.2017.03.016

[11] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *Int J Comput Vision* 88, 2 (2010), 303–338. DOI:https://doi.org/10.1007/s11263-009-0275-4

[12] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern Recogn Lett* 27, 8 (2006), 861–874. DOI:https://doi.org/10.1016/j.patrec.2005.10.010

[13] C. Ferri, J. Hernández-Orallo, and R. Modroiu. 2009. An experimental comparison of performance measures for classification. *Pattern Recogn Lett* 30, 1 (2009), 27–38. DOI:https://doi.org/10.1016/j.patrec.2008.08.010

[14] Alvaro Fuentes, Sook Yoon, Sang Cheol Kim, and Dong Sun Park. 2017. A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition. *Sensors* 17, 9 (2017), 2022. DOI:https://doi.org/10.3390/s17092022

[15] Geetharamani G. and Arun Pandian J. 2019. Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Comput Electr Eng* 76, (2019), 323–338. DOI:https://doi.org/10.1016/j.compeleceng.2019.04.011

[16] Aurélien Géron. 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Sebastopol, CA.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2017. *Deep Learning*. MIT Press, Cambridge, MA.

[18] Nadjib Guettari, Anne Sophie Capelle-Laizé, and Philippe Carré. 2016. Blind Image Steganalysis Based on Evidential K-Nearest Neighbors. *2016 Ieee Int Conf Image Process Icip* (2016), 2742–2746. DOI:https://doi.org/10.1109/icip.2016.7532858

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 Ieee Conf Comput Vis Pattern Recognit Cvpr* (2016), 770–778. DOI:https://doi.org/10.1109/cvpr.2016.90

[20] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian Error Linear Units (GELUs). *Arxiv* (2016).

[21] David P Hughes and Marcel Salathe. 2015. An open access repository of images on plant health to enable the development of mobile disease diagnostics. *Arxiv* (2015).

[22] Yusuke Kawasaki, Hiroyuki Uga, Satoshi Kagiwada, and Hitoshi Iyatomi. 2015. Advances in Visual Computing, 11th International Symposium, ISVC 2015, Las Vegas, NV, USA, December 14-16, 2015, Proceedings, Part II. *Lect Notes Comput Sc* (2015), 638–645. DOI:https://doi.org/10.1007/978-3-319-27863-6_59

[23] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp. *ICLR* (2017).

[24] Jan Kodovský, Jessica Fridrich, and Vojtěch Holub. 2012. Ensemble Classifiers for Steganalysis of Digital Media. *Ieee T Inf Foren Sec* 7, 2 (2012), 432–444. DOI:https://doi.org/10.1109/tifs.2011.2175919

[25] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. 2014. Efficient Mini-batch Training for Stochastic Optimization. *KDD '14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), 661–670. DOI:https://doi.org/10.1145/2623330.2623612

[26] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. 2019. Dying ReLU and Initialization: Theory and Numerical Examples. *Arxiv* (2019).

[27] George B. Lucas, C. Lee Campbell, and Leon T. Lucas. 1992. Introduction to Plant Diseases, Identification and Management. *undefined* (1992). DOI:https://doi.org/10.1007/978-1-4615-7294-7

[28] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *ICML* (2013). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/similar?doi=10.1.1.693.1422&type=ab

[29] Federico Martinelli, Riccardo Scalenghe, Salvatore Davino, Stefano Panno, Giuseppe Scuderi, Paolo Ruisi, Paolo Villa, Daniela Stroppiana, Mirco Boschetti, Luiz R. Goulart, Cristina E. Davis, and Abhaya M. Dandekar. 2015. Advanced methods of plant disease detection. A review. *Agron Sustain Dev* 35, 1 (2015), 1–25. DOI:https://doi.org/10.1007/s13593-014-0246-1

[30] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. 2016. Systematic evaluation of CNN advances on the ImageNet. *Arxiv* (2016). DOI:https://doi.org/10.1016/j.cviu.2017.05.007

[31] Sharada P. Mohanty, David P. Hughes, and Marcel Salathé. 2016. Using Deep Learning for Image-Based Plant Disease Detection. *Front Plant Sci* 7, (2016), 1419. DOI:https://doi.org/10.3389/fpls.2016.01419

[32] Sinno Jialin Pan and Qiang Yang. 2009. A Survey on Transfer Learning. *Ieee T Knowl Data En* 22, 10 (2009), 1345–1359. DOI:https://doi.org/10.1109/tkde.2009.191

[33] Ajeet Ram Pathak, Manjusha Pandey, and Siddharth Rautaray. 2018. Application of Deep Learning for Object Detection. *Procedia Comput Sci* 132, (2018), 1706–1717. DOI:https://doi.org/10.1016/j.procs.2018.05.144

[34] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *undefined*. Retrieved August 14, 2020 from https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants

[35] T. Rumpf, A.-K. Mahlein, U. Steiner, E.-C. Oerke, H.-W. Dehne, and L. Plümer. 2010. Early detection and classification of plant diseases with Support Vector Machines based on hyperspectral reflectance. *Comput Electron Agr* 74, 1 (2010), 91–99. DOI:https://doi.org/10.1016/j.compag.2010.06.009

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *Int J Comput Vision* 115, 3 (2015), 211–252. DOI:https://doi.org/10.1007/s11263-015-0816-y

[37] Jon Shlens. 2016. Google AI Blog: Train your own image classifier with Inception in TensorFlow. *undefined*. Retrieved January 17, 2021 from https://ai.googleblog.com/2016/03/train-your-own-image-classifier-with.html

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* (2014). Retrieved from https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[39] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. *Arxiv* (2014).

[40] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. *undefined*. Retrieved August 11, 2020 from https://arxiv.org/pdf/1512.00567.pdf

[41] WHO, IFAD, WFP, and UNICEF. 2020. *The State of Food Security and Nutrition in the World 2020*. FAO, IFAD, UNICEF, WFP and WHO. Retrieved from http://www.fao.org/documents/card/en/c/ca9692en

[42] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *Arxiv* (2015).

[43] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Arxiv* (2014).

[44] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2020. Dive into Deep Learning. *undefined*. Retrieved August 15, 2020 from https://d2l.ai/d2l-en.pdf

[45] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2016. Understanding deep learning requires rethinking generalization. *Arxiv* (2016).

[46] Zhilu Zhang and Mert R Sabuncu. 2018. Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels. *Arxiv* (2018).

[47] Barret Zoph and Quoc V Le. 2016. NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING. *Arxiv* (2016).

[48] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning Transferable Architectures for Scalable Image Recognition. *Arxiv* (2017).

Appendix A: Parameter Lists:

Parameter	Type	Description
α	Scalar	A scalar
$\boldsymbol{\alpha}$	Vector	A vector
$\boldsymbol{A}$	Matrix	A matrix
$\boldsymbol{A}_{i\times j}$	Matrix	A $i \times j$ matrix
a_i	Scalar	The ith entry of vector $\boldsymbol{\alpha}$
a_{ij}	Scalar	The (i, j) entry of Matrix **A**
$\boldsymbol{a}^{(n)}$	Vector	The nth vector in a dataset
h_θ	Function	The hypothesis function with parameter θ
$\boldsymbol{X} = \{\boldsymbol{x}^{(n)} \in R^d\}_{n=1}^{N}$	Set	Feature set, a set of d-dimensional vector $\boldsymbol{x}^{(n)} = [x_1^{(n)}, x_2^{(n)}, \dots, x_d^{(n)}]$ with n samples

$\boldsymbol{Y} = \{y^{(n)} \in R\}_{n=1}^{N}$	Set	Label set, a set of 1-dimentional scalar $y^{(n)}$ with n samples
$(\boldsymbol{x}^{(n)}, y^{(n)})$	Tuple	A data pair

Appendix B: Experimental Results

ResNet-50 with 0.2 validation split after 40 epochs:

```
====> Statistics: epochs=40, batch_size=32, validation_split=0.2, lr=0.001,
momentum=0.9, steps_per_epoch =100, feature shape= (256, 256, 3), no_classes=38,
loss_function=categorical_crossentropy
Epoch 1/40
1358/1358 - 1111s - loss: 3.7794 - accuracy: 0.0992 - precision: 0.0553 - recall:
5.2927e-04 - auc: 0.6495 - val_loss: 3.4739 - val_accuracy: 0.0938 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6477
Epoch 2/40
1358/1358 - 1108s - loss: 3.4177 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.6942 - val_loss: 3.3914 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6966
Epoch 3/40
1358/1358 - 1108s - loss: 3.3801 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.6963 - val_loss: 3.3708 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6999
Epoch 4/40
1358/1358 - 1099s - loss: 3.3664 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7005 - val_loss: 3.3615 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7015
Epoch 5/40
1358/1358 - 1151s - loss: 3.3595 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7017 - val_loss: 3.3563 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7024
Epoch 6/40
1358/1358 - 1171s - loss: 3.3553 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7024 - val_loss: 3.3529 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7030
Epoch 7/40
1358/1358 - 1222s - loss: 3.3525 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7028 - val_loss: 3.3506 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7034
Epoch 8/40
1358/1358 - 1240s - loss: 3.3506 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7032 - val_loss: 3.3491 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7035
Epoch 9/40
1358/1358 - 1198s - loss: 3.3493 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7032 - val_loss: 3.3479 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7035
Epoch 10/40
1358/1358 - 1168s - loss: 3.3483 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7031 - val_loss: 3.3471 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7033
Epoch 11/40
```

```
1358/1358 - 1239s - loss: 3.3476 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7030 - val_loss: 3.3464 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7034
Epoch 12/40
1358/1358 - 1433s - loss: 3.3470 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7031 - val_loss: 3.3459 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7034
Epoch 13/40
1358/1358 - 1347s - loss: 3.3466 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7031 - val_loss: 3.3455 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7034
Epoch 14/40
1358/1358 - 1317s - loss: 3.3462 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7029 - val_loss: 3.3452 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7032
Epoch 15/40
1358/1358 - 1287s - loss: 3.3460 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7029 - val_loss: 3.3450 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7033
Epoch 16/40
1358/1358 - 1273s - loss: 3.3458 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7028 - val_loss: 3.3448 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7033
Epoch 17/40
1358/1358 - 1293s - loss: 3.3456 - accuracy: 0.1011 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7032 - val_loss: 3.3446 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 18/40
1358/1358 - 1303s - loss: 3.3454 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3445 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 19/40
1358/1358 - 1325s - loss: 3.3453 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7034 - val_loss: 3.3444 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 20/40
1358/1358 - 1225s - loss: 3.3453 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7034 - val_loss: 3.3443 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 21/40
1358/1358 - 1192s - loss: 3.3452 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7034 - val_loss: 3.3442 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 22/40
1358/1358 - 1202s - loss: 3.3451 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3442 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 23/40
1358/1358 - 1182s - loss: 3.3450 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3441 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 24/40
1358/1358 - 1252s - loss: 3.3450 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3441 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 25/40
1358/1358 - 1262s - loss: 3.3449 - accuracy: 0.1003 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7034 - val_loss: 3.3440 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 26/40
```

```
1358/1358 - 1185s - loss: 3.3449 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7034 - val_loss: 3.3440 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 27/40
1358/1358 - 1151s - loss: 3.3449 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3440 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 28/40
1358/1358 - 1164s - loss: 3.3449 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7036 - val_loss: 3.3440 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 29/40
1358/1358 - 1136s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 30/40
1358/1358 - 1140s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 31/40
1358/1358 - 1147s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 32/40
1358/1358 - 1155s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 33/40
1358/1358 - 1180s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 34/40
1358/1358 - 1231s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7036 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7040
Epoch 35/40
1358/1358 - 1219s - loss: 3.3448 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3439 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 36/40
1358/1358 - 1148s - loss: 3.3447 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7036 - val_loss: 3.3438 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 37/40
1358/1358 - 1143s - loss: 3.3447 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3438 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 38/40
1358/1358 - 1133s - loss: 3.3447 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3438 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 39/40
1358/1358 - 1158s - loss: 3.3447 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7036 - val_loss: 3.3438 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
Epoch 40/40
1358/1358 - 1126s - loss: 3.3447 - accuracy: 0.1014 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7035 - val_loss: 3.3438 - val_accuracy: 0.1015 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7038
```

{'loss': [3.7794201374053955, 3.4177160263061523, 3.38014817237854,
3.3664169311523438, 3.3594608306884766, 3.3552613258361816, 3.352508783340454,
3.3506219387054443, 3.3492867946624756, 3.3483030796051025, 3.34757399559021,
3.3470206260681152, 3.3465828895568848, 3.3462467193603516, 3.345980405807495,
3.345770835876465, 3.3455991744995117, 3.3454482555389404, 3.345348596572876,
3.3452508449554443, 3.3451716899871826, 3.3451056480407715, 3.3450393676757812,
3.344996690750122, 3.3449394702911377, 3.3449175357818604, 3.34489369392395,
3.3448679447174072, 3.344839096069336, 3.3448121547698975, 3.3448078632354736,
3.344789505004883, 3.344780921936035, 3.3447635173797607, 3.344754695892334,
3.3447444438934326, 3.3447372913360596, 3.344717502593994, 3.3447229862213135,
3.344716787338257], 'accuracy': [0.09924981743097305, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10113678127527237, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.100308358669281, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286, 0.10138991475105286, 0.10138991475105286,
0.10138991475105286, 0.10138991475105286], 'precision': [0.0552884601056758, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0], 'recall': [0.0005292709683999419, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
'auc': [0.6495130658149719, 0.694237470626831, 0.6962851881980896, 0.7004990577697754,
0.7016860246658325, 0.7023895382881165, 0.7027557492256165, 0.7031845450401306,
0.7031943202018738, 0.703102946281433l, 0.703006386756897, 0.7030603289604187,
0.703079879283905, 0.7028968930244446, 0.7028513550758362, 0.7027973532676697,
0.7032042741775513, 0.7035109996795654, 0.7034026980400085, 0.7034397125244141,
0.7034322023391724, 0.7034515738487244, 0.7034856081008911, 0.703479528427124,
0.7034202218055725, 0.7033522129058838, 0.7034894227981567, 0.7036186456680298,
0.7035185694694519, 0.7034872174263, 0.7034769654273987, 0.7034669518470764,
0.7034639120101929, 0.7035550475120544, 0.70354825258255, 0.7036130428314209,
0.7035060524940491, 0.7034614086151123, 0.7035642862319946, 0.7035232782363892],
'val_loss': [3.473935842514038, 3.39143443107605, 3.3708136081695557,
3.361534357070923, 3.3562769889831543, 3.3529176712036133, 3.3506462574005127,
3.349062204360962, 3.3479080200195312, 3.3470656871795654, 3.346423864364624,
3.345928430557251, 3.345547676086426, 3.345242738723755, 3.345004081726074,
3.3448078632354736, 3.3446462154388428, 3.344517946243286, 3.344406843185425,
3.3443186283111572, 3.344245433807373, 3.3441812992095947, 3.3441295623779297,
3.344087839126587, 3.344048500061035, 3.344014883041382, 3.3439807891845703,
3.3439581394195557, 3.343935012817383, 3.3439223766326904, 3.3439011573791504,
3.3438875675201416, 3.343873977661133, 3.343860149383545, 3.3438522815704346,
3.343841075897217, 3.3438334465026855, 3.3438363075256348, 3.3438191413879395,
3.343811273574829], 'val_accuracy': [0.09383353590965271, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141, 0.10148400813341141, 0.10148400813341141,
0.10148400813341141, 0.10148400813341141], 'val_precision': [0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0], 'val_recall': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}, 'val_auc': [0.647680938243866,
0.6965564489364624, 0.6998650431632996, 0.7015079259872437, 0.7024268507957458,
0.7030341029167175, 0.7033684253692627, 0.7034826874732971, 0.7034728527069092,
0.7033368945121765, 0.7033883929252625, 0.7033968567848206, 0.7033962607383728,
0.7031619548797607, 0.7033458352088928, 0.7033438086509705, 0.7037695050239563,
0.7037695050239563, 0.7037691473960876, 0.7037688493728638, 0.7037688493728638,
0.7037685513496399, 0.7037685513496399, 0.7037690877914429, 0.7037685513496399,
0.7038388848304749, 0.7038385272026062, 0.7038385272026062, 0.7038388848304749,
0.7038388252258301, 0.7038388252258301, 0.7038390636444092, 0.7038381695747375,
0.7039579749107361, 0.7038381695747375, 0.7038388252258301, 0.7038385272026062,
0.7038005590438843, 0.7038385272026062, 0.7038385272026062]}

InceptionV3 with 0.2 validation split after 40 epochs:

1358/1358 - 1142s - loss: 0.4532 - accuracy: 0.8766 - precision: 0.9475 - recall:
0.8204 - auc: 0.9950 - val_loss: 0.1648 - val_accuracy: 0.9522 - val_precision: 0.9665
- val_recall: 0.9337 - val_auc: 0.9994
Epoch 2/40
1358/1358 - 1144s - loss: 0.0897 - accuracy: 0.9774 - precision: 0.9839 - recall:
0.9696 - auc: 0.9998 - val_loss: 0.1086 - val_accuracy: 0.9671 - val_precision: 0.9755
- val_recall: 0.9587 - val_auc: 0.9997
Epoch 3/40
1358/1358 - 1119s - loss: 0.0421 - accuracy: 0.9926 - precision: 0.9948 - recall:
0.9901 - auc: 0.9999 - val_loss: 0.1107 - val_accuracy: 0.9639 - val_precision: 0.9713
- val_recall: 0.9570 - val_auc: 0.9995
Epoch 4/40
1358/1358 - 1117s - loss: 0.0224 - accuracy: 0.9981 - precision: 0.9988 - recall:
0.9972 - auc: 1.0000 - val_loss: 0.0814 - val_accuracy: 0.9763 - val_precision: 0.9801
- val_recall: 0.9703 - val_auc: 0.9996
Epoch 5/40
1358/1358 - 1121s - loss: 0.0145 - accuracy: 0.9993 - precision: 0.9995 - recall:
0.9990 - auc: 1.0000 - val_loss: 0.0901 - val_accuracy: 0.9707 - val_precision: 0.9747
- val_recall: 0.9661 - val_auc: 0.9994
Epoch 6/40
1358/1358 - 1149s - loss: 0.0102 - accuracy: 0.9999 - precision: 0.9999 - recall:
0.9998 - auc: 1.0000 - val_loss: 0.0714 - val_accuracy: 0.9781 - val_precision: 0.9820
- val_recall: 0.9737 - val_auc: 0.9997
Epoch 7/40
1358/1358 - 1173s - loss: 0.0078 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0698 - val_accuracy: 0.9776 - val_precision: 0.9818
- val_recall: 0.9740 - val_auc: 0.9996
Epoch 8/40
1358/1358 - 1140s - loss: 0.0064 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0693 - val_accuracy: 0.9772 - val_precision: 0.9811
- val_recall: 0.9735 - val_auc: 0.9997
Epoch 9/40
1358/1358 - 1102s - loss: 0.0053 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0687 - val_accuracy: 0.9783 - val_precision: 0.9816
- val_recall: 0.9751 - val_auc: 0.9996
Epoch 10/40
1358/1358 - 1105s - loss: 0.0047 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0659 - val_accuracy: 0.9790 - val_precision: 0.9825
- val_recall: 0.9756 - val_auc: 0.9995
Epoch 11/40
1358/1358 - 1101s - loss: 0.0041 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0651 - val_accuracy: 0.9794 - val_precision: 0.9827
- val_recall: 0.9764 - val_auc: 0.9995
Epoch 12/40

```
1358/1358 - 1090s - loss: 0.0037 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0644 - val_accuracy: 0.9792 - val_precision: 0.9821
- val_recall: 0.9758 - val_auc: 0.9996
Epoch 13/40
1358/1358 - 1097s - loss: 0.0033 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0654 - val_accuracy: 0.9790 - val_precision: 0.9820
- val_recall: 0.9766 - val_auc: 0.9995
Epoch 14/40
1358/1358 - 1096s - loss: 0.0031 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0643 - val_accuracy: 0.9794 - val_precision: 0.9824
- val_recall: 0.9768 - val_auc: 0.9995
Epoch 15/40
1358/1358 - 1100s - loss: 0.0028 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0639 - val_accuracy: 0.9789 - val_precision: 0.9818
- val_recall: 0.9760 - val_auc: 0.9995
Epoch 16/40
1358/1358 - 1100s - loss: 0.0026 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0639 - val_accuracy: 0.9791 - val_precision: 0.9817
- val_recall: 0.9766 - val_auc: 0.9994
Epoch 17/40
1358/1358 - 1112s - loss: 0.0024 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0635 - val_accuracy: 0.9794 - val_precision: 0.9821
- val_recall: 0.9767 - val_auc: 0.9995
Epoch 18/40
1358/1358 - 1095s - loss: 0.0023 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0636 - val_accuracy: 0.9796 - val_precision: 0.9817
- val_recall: 0.9766 - val_auc: 0.9995
Epoch 19/40
1358/1358 - 1107s - loss: 0.0021 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0634 - val_accuracy: 0.9797 - val_precision: 0.9820
- val_recall: 0.9775 - val_auc: 0.9995
Epoch 20/40
1358/1358 - 1116s - loss: 0.0020 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0625 - val_accuracy: 0.9799 - val_precision: 0.9829
- val_recall: 0.9773 - val_auc: 0.9995
Epoch 21/40
1358/1358 - 1121s - loss: 0.0019 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0625 - val_accuracy: 0.9809 - val_precision: 0.9828
- val_recall: 0.9778 - val_auc: 0.9994
Epoch 22/40
1358/1358 - 1099s - loss: 0.0018 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0623 - val_accuracy: 0.9797 - val_precision: 0.9824
- val_recall: 0.9776 - val_auc: 0.9994
Epoch 23/40
1358/1358 - 1089s - loss: 0.0017 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0626 - val_accuracy: 0.9797 - val_precision: 0.9821
- val_recall: 0.9777 - val_auc: 0.9994
Epoch 24/40
1358/1358 - 1095s - loss: 0.0016 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0626 - val_accuracy: 0.9795 - val_precision: 0.9823
- val_recall: 0.9774 - val_auc: 0.9994
Epoch 25/40
1358/1358 - 1121s - loss: 0.0015 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0625 - val_accuracy: 0.9800 - val_precision: 0.9821
- val_recall: 0.9774 - val_auc: 0.9994
Epoch 26/40
1358/1358 - 1110s - loss: 0.0015 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0624 - val_accuracy: 0.9796 - val_precision: 0.9819
- val_recall: 0.9779 - val_auc: 0.9994
Epoch 27/40
```

```
1358/1358 - 1104s - loss: 0.0014 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0624 - val_accuracy: 0.9796 - val_precision: 0.9823
- val_recall: 0.9775 - val_auc: 0.9994
Epoch 28/40
1358/1358 - 1119s - loss: 0.0014 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0616 - val_accuracy: 0.9803 - val_precision: 0.9820
- val_recall: 0.9778 - val_auc: 0.9994
Epoch 29/40
1358/1358 - 1120s - loss: 0.0013 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0619 - val_accuracy: 0.9798 - val_precision: 0.9822
- val_recall: 0.9778 - val_auc: 0.9994
Epoch 30/40
1358/1358 - 1112s - loss: 0.0013 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0620 - val_accuracy: 0.9800 - val_precision: 0.9822
- val_recall: 0.9781 - val_auc: 0.9994
Epoch 31/40
1358/1358 - 1124s - loss: 0.0012 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0614 - val_accuracy: 0.9801 - val_precision: 0.9824
- val_recall: 0.9776 - val_auc: 0.9994
Epoch 32/40
1358/1358 - 1125s - loss: 0.0012 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0617 - val_accuracy: 0.9805 - val_precision: 0.9826
- val_recall: 0.9780 - val_auc: 0.9994
Epoch 33/40
1358/1358 - 1099s - loss: 0.0011 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0619 - val_accuracy: 0.9801 - val_precision: 0.9825
- val_recall: 0.9784 - val_auc: 0.9994
Epoch 34/40
1358/1358 - 1078s - loss: 0.0011 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0618 - val_accuracy: 0.9806 - val_precision: 0.9827
- val_recall: 0.9782 - val_auc: 0.9994
Epoch 35/40
1358/1358 - 1374s - loss: 0.0011 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0612 - val_accuracy: 0.9802 - val_precision: 0.9823
- val_recall: 0.9780 - val_auc: 0.9994
Epoch 36/40
1358/1358 - 1122s - loss: 0.0010 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0616 - val_accuracy: 0.9805 - val_precision: 0.9822
- val_recall: 0.9782 - val_auc: 0.9994
Epoch 37/40
1358/1358 - 1115s - loss: 9.8997e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0616 - val_accuracy: 0.9800 - val_precision: 0.9823
- val_recall: 0.9782 - val_auc: 0.9993
Epoch 38/40
1358/1358 - 1115s - loss: 9.6077e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0613 - val_accuracy: 0.9807 - val_precision: 0.9825
- val_recall: 0.9783 - val_auc: 0.9994
Epoch 39/40
1358/1358 - 1128s - loss: 9.3324e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0613 - val_accuracy: 0.9805 - val_precision: 0.9821
- val_recall: 0.9783 - val_auc: 0.9994
Epoch 40/40
1358/1358 - 1150s - loss: 9.0539e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - auc: 1.0000 - val_loss: 0.0614 - val_accuracy: 0.9811 - val_precision: 0.9825
- val_recall: 0.9787 - val_auc: 0.9994
{'loss': [0.45319825410842896, 0.08965033292770386, 0.04208862781524658,
0.022353066131472588, 0.014545547775924206, 0.010241761803627014,
0.007815362885594368, 0.006361474748700857, 0.005339575000107288,
0.004656187258660793, 0.004111648071557283, 0.003707831958308816,
0.0033438564278185368, 0.0030651011038571596, 0.002813779516145587,
```

0.002607632428407669, 0.0024247164838016033, 0.002264973009005189,
0.0021261507645249367, 0.0020210779222473502, 0.0018922224408015609,
0.0017943409038707614, 0.0017064287094399333, 0.0016240711556747556,
0.001547843450680375, 0.0014801520155742764, 0.0014174151001498103,
0.001359714544378221, 0.0013072560541331768, 0.0012550319079309702,
0.0012090313248336315, 0.0011670617386698723, 0.0011276894947513938,
0.0010881659109145403, 0.0010549064027145505, 0.001019268180243671,
0.0009899697033688426, 0.00096077227499336, 0.0009332436020486057,
0.0009053936810232699], 'accuracy': [0.8766338229179382, 0.9774484634399414,
0.9926362037658691, 0.9980900287628174, 0.9993326663970947, 0.9998619556427002,
0.9999539852142334, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0], 'precision': [0.9475097060203552, 0.9839109182357788,
0.9947515726089478, 0.9988244771957397, 0.999516487121582, 0.9998849034309387,
0.9999539852142334, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0], 'recall': [0.8203930258750916, 0.9696014523506165, 0.9900588989257812,
0.997169554233551, 0.9990105032920837, 0.9997698664665222, 0.9999539852142334, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
'auc': [0.9949777722358704, 0.9997520446777344, 0.9999417066574097, 0.9999861121177673,
0.9999997019767761, 1.0000001192092896, 0.9999999403953552, 0.9999999403953552,
0.9999999403953552, 1.0, 1.0, 1.0, 0.9999999403953552, 1.0, 1.0, 1.0000001192092896,
1.0, 0.9999999403953552, 1.0, 0.9999999403953552, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.9999999403953552, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0],
'val_loss': [0.16482393443584442, 0.10861088335514069, 0.11070862412452698,
0.08141910284757614, 0.09012922644615173, 0.07139179855585098, 0.06975626945495605,
0.06929273903369904, 0.06870284676551819, 0.06593102961778641, 0.06514790654182434,
0.06442967057228088, 0.06539317220449448, 0.0642584040760994, 0.0639205351471901,
0.06385207921266556, 0.06354766339063644, 0.0636015310883522, 0.06341460347175598,
0.0625079870223999, 0.06253274530172348, 0.062263425439596176, 0.06260053813457489,
0.06258176267147064, 0.062464162707328796, 0.06243719160556793, 0.06237777695059776,
0.06160534545779228, 0.06185286492109299, 0.062020029872655587, 0.061350952833890915,
0.061738818883895874, 0.06186292693018913, 0.06176263839006424, 0.06121692806482315,
0.06163749098777771, 0.061648573726415634, 0.061270732432603836, 0.061316609382629395,
0.06135638430714607], 'val_accuracy': [0.9521614909172058, 0.9670937657356262,
0.9638676643371582, 0.976311206817627, 0.97068852186203, 0.9780625104904175,
0.9776016473770142, 0.9772329330444336, 0.9783390164375305, 0.9789842367172241,
0.9793529510498047, 0.9791685938835144, 0.9789842367172241, 0.9794450998306274,
0.9788920879364014, 0.9790763854980469, 0.9794450998306274, 0.9796294569969177,
0.9797216057777405, 0.9799059629440308, 0.9809198975563049, 0.9797216057777405,
0.9797216057777405, 0.979537308216095, 0.9799981713294983, 0.9796294569969177,
0.9796294569969177, 0.9802746772766113, 0.979813814163208, 0.9799981713294983,
0.980090320110321, 0.9804590344429016, 0.980090320110321, 0.9805511832237244,
0.9801825284957886, 0.9804590344429016, 0.9799981713294983, 0.9807355403900146,
0.9804590344429016, 0.9811042547225952], 'val_precision': [0.9665108323097229,
0.9755205512046814, 0.9712815880775452, 0.9800763726234436, 0.9747047424316406,
0.9819669127464294, 0.9817894697189331, 0.9810514450073242, 0.9816275238990784,
0.9824561476707458, 0.9827442169189453, 0.9820948243141174, 0.9820187091827393,
0.9823862314224243, 0.9818266034126282, 0.981745719909668, 0.982111394405365,
0.981745719909668, 0.9820353984832764, 0.9828513264656067, 0.982768177986145,
0.9824008941650391, 0.9821296334266663, 0.9823065996170044, 0.9821246862411499,
0.9818602204322815, 0.9823082685470581, 0.9820403456687927, 0.9822221994400024,
0.9822271466255188, 0.9824008941650391, 0.9825893640518188, 0.9825064539909363,
0.9826852083206177, 0.9823164343833923, 0.9822288155555725, 0.9823213815689087,
0.9825048446655273, 0.9821411967277527, 0.9825113415718079], 'val_recall':
[0.9337266087532043, 0.9587058424949646, 0.9570467472076416, 0.9703198671340942,
0.9660798311233521, 0.9737303256988525, 0.9740068316459656, 0.9735459685325623,
0.9751129150390625, 0.9755737781524658, 0.9764033555984497, 0.9757581353187561,
0.97658771276474, 0.9767720699310303, 0.9760346412658691, 0.97658771276474,

0.9766798615455627, 0.97658771276474, 0.9775094389915466, 0.9773250818252563,
0.9777859449386597, 0.9776016473770142, 0.9776937961578369, 0.9774172902107239,
0.9774172902107239, 0.9778781533241272, 0.9775094389915466, 0.9777859449386597,
0.9777859449386597, 0.9780625104904175, 0.9776016473770142, 0.97797030210495,
0.9784311652183533, 0.9782468676567078, 0.97797030210495, 0.9781546592712402,
0.9782468676567078, 0.9783390164375305, 0.9783390164375305, 0.9787077307701111],
'val_auc': [0.9994434714317322, 0.9996668696403503, 0.9995075464248657,
0.9996349811553955, 0.999427080154419, 0.9996538758277893, 0.9996108412742615,
0.9997454881668091, 0.9995660185813904, 0.9995242953300476, 0.9995262622833252,
0.99957275390625, 0.9994784593582153, 0.9994813203811646, 0.999479353427887,
0.9994348287582397, 0.9994809627532959, 0.9994810819625854, 0.9994816184043884,
0.9994829297065735, 0.9994367957115173, 0.9994379281997681, 0.9994366765022278,
0.9994368553161621, 0.9993918538093567, 0.9994365572929382, 0.9993916153907776,
0.9994379281997681, 0.9993922114372253, 0.9993886351585388, 0.9994396567344666,
0.999436616897583, 0.9994363784790039, 0.9993910193443298, 0.9993923306465149,
0.9993897080421448, 0.9993460178375244, 0.9993913769721985, 0.9993903636932373,
0.9993903040885925]}

NASNetMobile with 0.2 validation split after 40 epochs:

====> Statistics: MODEL_NAME=NASNetMobile, epochs=40, batch_size=32,
validation_split=0.2, lr=0.001, momentum=0.9, steps_per_epoch =10, feature shape=
(256, 256, 3), no_classes=38, loss_function=categorical_crossentropy
Epoch 1/40
1358/1358 - 1175s - loss: 3.5011 - accuracy: 0.1505 - precision: 0.5734 - recall:
0.0318 - auc: 0.7444 - val_loss: 2.8570 - val_accuracy: 0.2357 - val_precision: 0.5597
- val_recall: 0.0825 - val_auc: 0.8076
Epoch 2/40
1358/1358 - 1206s - loss: 2.8436 - accuracy: 0.2218 - precision: 0.6614 - recall:
0.0802 - auc: 0.8148 - val_loss: 2.7323 - val_accuracy: 0.2542 - val_precision: 0.7219
- val_recall: 0.0373 - val_auc: 0.8383
Epoch 3/40
1358/1358 - 1185s - loss: 2.7535 - accuracy: 0.2438 - precision: 0.6855 - recall:
0.0975 - auc: 0.8339 - val_loss: 2.6583 - val_accuracy: 0.2600 - val_precision: 0.7050
- val_recall: 0.1152 - val_auc: 0.8523
Epoch 4/40
1358/1358 - 1157s - loss: 2.7216 - accuracy: 0.2480 - precision: 0.6894 - recall:
0.1040 - auc: 0.8404 - val_loss: 2.6895 - val_accuracy: 0.2584 - val_precision: 0.9327
- val_recall: 0.0575 - val_auc: 0.8492
Epoch 5/40
1358/1358 - 1148s - loss: 2.6627 - accuracy: 0.2624 - precision: 0.7174 - recall:
0.1115 - auc: 0.8496 - val_loss: 2.6542 - val_accuracy: 0.2645 - val_precision: 0.6486
- val_recall: 0.0885 - val_auc: 0.8573
Epoch 6/40

```
1358/1358 - 1164s - loss: 2.6506 - accuracy: 0.2639 - precision: 0.7109 - recall:
0.1136 - auc: 0.8517 - val_loss: 2.5558 - val_accuracy: 0.2749 - val_precision: 0.7125
- val_recall: 0.1590 - val_auc: 0.8663
Epoch 7/40
1358/1358 - 1132s - loss: 2.6119 - accuracy: 0.2695 - precision: 0.7151 - recall:
0.1184 - auc: 0.8570 - val_loss: 2.5286 - val_accuracy: 0.2767 - val_precision: 0.7114
- val_recall: 0.1163 - val_auc: 0.8704
Epoch 8/40
1358/1358 - 1127s - loss: 2.5879 - accuracy: 0.2750 - precision: 0.7084 - recall:
0.1241 - auc: 0.8606 - val_loss: 2.5031 - val_accuracy: 0.2899 - val_precision: 0.7494
- val_recall: 0.1472 - val_auc: 0.8747
Epoch 9/40
1358/1358 - 1210s - loss: 2.5824 - accuracy: 0.2753 - precision: 0.7252 - recall:
0.1237 - auc: 0.8606 - val_loss: 2.4860 - val_accuracy: 0.2913 - val_precision: 0.8178
- val_recall: 0.1361 - val_auc: 0.8756
Epoch 10/40
1358/1358 - 1137s - loss: 2.5605 - accuracy: 0.2802 - precision: 0.7378 - recall:
0.1271 - auc: 0.8642 - val_loss: 2.4744 - val_accuracy: 0.2822 - val_precision: 0.7780
- val_recall: 0.1308 - val_auc: 0.8770
Epoch 11/40
1358/1358 - 1131s - loss: 2.5352 - accuracy: 0.2849 - precision: 0.7342 - recall:
0.1328 - auc: 0.8673 - val_loss: 2.5311 - val_accuracy: 0.2889 - val_precision: 0.8044
- val_recall: 0.1433 - val_auc: 0.8629
Epoch 12/40
1358/1358 - 1118s - loss: 2.5507 - accuracy: 0.2838 - precision: 0.7218 - recall:
0.1302 - auc: 0.8644 - val_loss: 2.4688 - val_accuracy: 0.2987 - val_precision: 0.6464
- val_recall: 0.1599 - val_auc: 0.8781
Epoch 13/40
1358/1358 - 1129s - loss: 2.4999 - accuracy: 0.2975 - precision: 0.7297 - recall:
0.1433 - auc: 0.8709 - val_loss: 2.6593 - val_accuracy: 0.2694 - val_precision: 0.9317
- val_recall: 0.0591 - val_auc: 0.8464
Epoch 14/40
1358/1358 - 1121s - loss: 2.4694 - accuracy: 0.3062 - precision: 0.7211 - recall:
0.1499 - auc: 0.8743 - val_loss: 2.3280 - val_accuracy: 0.3321 - val_precision: 0.7661
- val_recall: 0.1690 - val_auc: 0.8937
Epoch 15/40
1358/1358 - 1107s - loss: 2.4336 - accuracy: 0.3138 - precision: 0.7150 - recall:
0.1548 - auc: 0.8785 - val_loss: 2.3270 - val_accuracy: 0.3446 - val_precision: 0.8288
- val_recall: 0.1517 - val_auc: 0.8926
Epoch 16/40
```

```
1358/1358 - 1099s - loss: 2.3917 - accuracy: 0.3219 - precision: 0.7061 - recall:
0.1558 - auc: 0.8842 - val_loss: 2.3743 - val_accuracy: 0.3261 - val_precision: 0.7104
- val_recall: 0.1757 - val_auc: 0.8848
Epoch 17/40
1358/1358 - 1096s - loss: 2.3442 - accuracy: 0.3319 - precision: 0.7074 - recall:
0.1609 - auc: 0.8902 - val_loss: 2.3490 - val_accuracy: 0.3262 - val_precision: 0.6661
- val_recall: 0.2035 - val_auc: 0.8897
Epoch 18/40
1358/1358 - 1106s - loss: 2.3372 - accuracy: 0.3312 - precision: 0.7009 - recall:
0.1618 - auc: 0.8910 - val_loss: 2.6864 - val_accuracy: 0.2365 - val_precision: 0.7183
- val_recall: 0.0799 - val_auc: 0.8413
Epoch 19/40
1358/1358 - 1099s - loss: 2.3315 - accuracy: 0.3334 - precision: 0.6942 - recall:
0.1632 - auc: 0.8916 - val_loss: 2.2188 - val_accuracy: 0.3745 - val_precision: 0.7639
- val_recall: 0.1309 - val_auc: 0.9063
Epoch 20/40
1358/1358 - 1096s - loss: 2.3264 - accuracy: 0.3349 - precision: 0.6891 - recall:
0.1657 - auc: 0.8923 - val_loss: 2.1921 - val_accuracy: 0.3615 - val_precision: 0.7560
- val_recall: 0.2042 - val_auc: 0.9063
Epoch 21/40
1358/1358 - 1110s - loss: 2.2834 - accuracy: 0.3460 - precision: 0.7016 - recall:
0.1719 - auc: 0.8971 - val_loss: 2.3576 - val_accuracy: 0.3302 - val_precision: 0.7326
- val_recall: 0.1513 - val_auc: 0.8878
Epoch 22/40
1358/1358 - 1092s - loss: 2.2866 - accuracy: 0.3422 - precision: 0.6946 - recall:
0.1752 - auc: 0.8968 - val_loss: 2.4685 - val_accuracy: 0.3104 - val_precision: 0.7912
- val_recall: 0.0943 - val_auc: 0.8779
Epoch 23/40
1358/1358 - 1117s - loss: 2.2954 - accuracy: 0.3443 - precision: 0.6992 - recall:
0.1743 - auc: 0.8950 - val_loss: 2.1557 - val_accuracy: 0.3740 - val_precision: 0.7891
- val_recall: 0.1794 - val_auc: 0.9121
Epoch 24/40
1358/1358 - 1149s - loss: 2.2518 - accuracy: 0.3507 - precision: 0.6978 - recall:
0.1835 - auc: 0.9006 - val_loss: 2.2991 - val_accuracy: 0.3368 - val_precision: 0.5563
- val_recall: 0.2136 - val_auc: 0.8971
Epoch 25/40
1358/1358 - 1270s - loss: 2.2564 - accuracy: 0.3531 - precision: 0.7039 - recall:
0.1826 - auc: 0.8995 - val_loss: 2.3299 - val_accuracy: 0.3309 - val_precision: 0.6633
- val_recall: 0.1831 - val_auc: 0.8909
Epoch 26/40
```

```
1358/1358 - 1222s - loss: 2.2436 - accuracy: 0.3538 - precision: 0.7005 - recall:
0.1849 - auc: 0.9015 - val_loss: 2.1796 - val_accuracy: 0.3682 - val_precision: 0.7404
- val_recall: 0.2337 - val_auc: 0.9048
Epoch 27/40
1358/1358 - 1107s - loss: 2.2147 - accuracy: 0.3644 - precision: 0.7137 - recall:
0.1925 - auc: 0.9040 - val_loss: 2.2063 - val_accuracy: 0.3679 - val_precision: 0.7626
- val_recall: 0.1863 - val_auc: 0.9029
Epoch 28/40
1358/1358 - 1149s - loss: 2.2103 - accuracy: 0.3634 - precision: 0.7102 - recall:
0.1966 - auc: 0.9042 - val_loss: 2.1054 - val_accuracy: 0.3825 - val_precision: 0.7859
- val_recall: 0.1999 - val_auc: 0.9178
Epoch 29/40
1358/1358 - 1107s - loss: 2.1955 - accuracy: 0.3689 - precision: 0.7173 - recall:
0.2065 - auc: 0.9051 - val_loss: 2.4087 - val_accuracy: 0.3467 - val_precision: 0.6547
- val_recall: 0.2378 - val_auc: 0.8763
Epoch 30/40
1358/1358 - 1117s - loss: 2.1898 - accuracy: 0.3743 - precision: 0.7229 - recall:
0.2162 - auc: 0.9057 - val_loss: 2.2224 - val_accuracy: 0.3613 - val_precision: 0.7219
- val_recall: 0.2098 - val_auc: 0.9043
Epoch 31/40
1358/1358 - 1116s - loss: 2.1958 - accuracy: 0.3724 - precision: 0.7273 - recall:
0.2193 - auc: 0.9041 - val_loss: 2.0842 - val_accuracy: 0.4090 - val_precision: 0.7917
- val_recall: 0.2614 - val_auc: 0.9130
Epoch 32/40
1358/1358 - 1116s - loss: 2.1784 - accuracy: 0.3783 - precision: 0.7301 - recall:
0.2252 - auc: 0.9061 - val_loss: 2.3708 - val_accuracy: 0.3456 - val_precision: 0.6265
- val_recall: 0.2054 - val_auc: 0.8853
Epoch 33/40
1358/1358 - 1110s - loss: 2.1532 - accuracy: 0.3878 - precision: 0.7439 - recall:
0.2355 - auc: 0.9085 - val_loss: 2.1516 - val_accuracy: 0.3882 - val_precision: 0.7392
- val_recall: 0.2119 - val_auc: 0.9107
Epoch 34/40
1358/1358 - 1120s - loss: 2.1562 - accuracy: 0.3826 - precision: 0.7327 - recall:
0.2343 - auc: 0.9084 - val_loss: 2.3538 - val_accuracy: 0.3320 - val_precision: 0.8044
- val_recall: 0.1634 - val_auc: 0.8833
Epoch 35/40
1358/1358 - 1123s - loss: 2.1452 - accuracy: 0.3890 - precision: 0.7374 - recall:
0.2372 - auc: 0.9094 - val_loss: 2.0820 - val_accuracy: 0.3826 - val_precision: 0.7287
- val_recall: 0.2723 - val_auc: 0.9160
Epoch 36/40
```

```
1358/1358 - 1139s - loss: 2.1260 - accuracy: 0.3924 - precision: 0.7501 - recall:
0.2423 - auc: 0.9111 - val_loss: 2.0707 - val_accuracy: 0.3978 - val_precision: 0.7371
- val_recall: 0.2579 - val_auc: 0.9177
Epoch 37/40
1358/1358 - 1175s - loss: 2.1209 - accuracy: 0.3948 - precision: 0.7505 - recall:
0.2451 - auc: 0.9116 - val_loss: 2.2674 - val_accuracy: 0.3632 - val_precision: 0.7156
- val_recall: 0.2289 - val_auc: 0.8945
Epoch 38/40
1358/1358 - 1138s - loss: 2.1034 - accuracy: 0.4017 - precision: 0.7552 - recall:
0.2517 - auc: 0.9128 - val_loss: 1.9564 - val_accuracy: 0.4330 - val_precision: 0.8452
- val_recall: 0.2668 - val_auc: 0.9280
Epoch 39/40
1358/1358 - 1138s - loss: 2.0925 - accuracy: 0.4008 - precision: 0.7577 - recall:
0.2530 - auc: 0.9144 - val_loss: 2.0623 - val_accuracy: 0.4153 - val_precision: 0.7889
- val_recall: 0.2673 - val_auc: 0.9150
Epoch 40/40
1358/1358 - 1140s - loss: 2.0956 - accuracy: 0.4040 - precision: 0.7670 - recall:
0.2547 - auc: 0.9135 - val_loss: 2.0632 - val_accuracy: 0.3996 - val_precision: 0.8140
- val_recall: 0.2456 - val_auc: 0.9200
2020-12-21 14:43:47.747367: W tensorflow/python/util/util.cc:348] Sets are not
currently considered sequences, but this may change in the future, so consider
avoiding using them.
{'loss': [3.5010833740234375, 2.843637704849243, 2.7535059452056885,
2.7215726375579834, 2.6627230644226074, 2.650569438934326, 2.6119086742401123,
2.587904930114746, 2.5824472904205322, 2.5604727268218994, 2.535235643386841,
2.5506908893585205, 2.499910354614258, 2.46936297416687, 2.433619499206543,
2.391732692718506, 2.3442251682281494, 2.3371691703796387, 2.331547975540161,
2.326397180557251, 2.2833731174468994, 2.28657603263855, 2.295403480529785,
2.2518467903137207, 2.2563929557800293, 2.2436439990997314, 2.2147254943847656,
2.2102997303009033, 2.1955056190490723, 2.1898255348205566, 2.195788860321045,
2.1784276962280273, 2.1531789302825928, 2.1561641693115234, 2.145231008529663,
2.126011610031128, 2.1209323406219482, 2.1034393310546875, 2.092529535293579,
2.095607280731201], 'accuracy': [0.15047404170036316, 0.22178755700588226,
0.24383284151554108, 0.2480439990758896, 0.26244938373565674, 0.2639221251010895,
0.2694909870624542, 0.2749677896499634, 0.27526694536209106, 0.28023749589920044,
0.28488585352897644, 0.2838273048400879, 0.29749631881713867, 0.3062407970428467,
0.31383469700813293, 0.32188880443573, 0.33192193508148193, 0.33123159408569336,
0.33341771364212036, 0.3349134624004364, 0.3460051417350769, 0.34218519926071167,
0.34434831142425537, 0.3507455885410309, 0.3531157970428467, 0.35382917523384094,
0.3644145727157593, 0.3633560240268707, 0.36885586380958557, 0.3743326663970947,
```

0.3723996579647064, 0.37826767563819885, 0.3878405690193176, 0.38263991475105286,
0.3889681398868561, 0.3923508822917938, 0.3947901427745819, 0.4016936719417572,
0.40075019001960754, 0.4039718210697174], 'precision': [0.573383092880249,
0.6614187955856323, 0.6855478286743164, 0.6894447803497314, 0.7173945307731628,
0.710871160030365, 0.715119481086731, 0.7084099650382996, 0.7251517176628113,
0.7378472089767456, 0.7341643571853638, 0.7218112349510193, 0.7296695709228516,
0.7210718393325806, 0.7150452136993408, 0.7061216235160828, 0.70744788646698,
0.7009271383285522, 0.6941970586776733, 0.6890547275543213, 0.7016439437866211,
0.6946091651916504, 0.6992245316505432, 0.6977598667144775, 0.7038679718971252,
0.7004708647727966, 0.7137371897697449, 0.710224449634552, 0.7172540426254272,
0.7228804230690002, 0.7272588610649109, 0.7301149368286133, 0.7438767552375793,
0.7327430844306946, 0.7373542189598083, 0.7500534057617188, 0.7504756450653076,
0.7551597952842712, 0.7576843500137329, 0.7670292854309082], 'recall':
[0.031825292855501175, 0.0802420824766159, 0.09747790545225143, 0.10401325672864914,
0.11151509732007980, 0.11360916495323181, 0.1184186339378357, 0.12405651807785034,
0.12373435497283936, 0.1271400898694992, 0.13282400369644165, 0.13022367656230927,
0.14329436421394348, 0.1498527228832245, 0.15475423634052277, 0.15581277012825012,
0.1608753651380539, 0.16179583966732025, 0.16324558854103088, 0.16573084890842438,
0.171875, 0.17523472011089325, 0.1742912381887436, 0.18349595367908478,
0.18257547914981842, 0.18485364317893982, 0.19249355792999268, 0.1966126710176468,
0.20653074979782104, 0.2162187099456787, 0.21930228173732758, 0.2251702845096588,
0.23552559316158295, 0.2342599481344223, 0.23715943098068237, 0.24231407046318054,
0.24507547914981842, 0.2517488896846771, 0.25299152731895447, 0.2547174096107483],
'auc': [0.7444429397583008, 0.814803421497345, 0.833871066570282, 0.8404320478439331,
0.8496297001838684, 0.8516793251037598, 0.8569921255111694, 0.8605955243110657,
0.8605504631996155, 0.8642175793647766, 0.8672531843185425, 0.8644254207611084,
0.8709063529968262, 0.8743382096290588, 0.8785164952278137, 0.8842132687568665,
0.8902016282081604, 0.890958309173584, 0.8915610313415527, 0.8922507762908936,
0.8971002697944641, 0.8967610001564026, 0.895012378692627, 0.9005528688430786,
0.8995325565338135, 0.9014968276023865, 0.9039731025695801, 0.904184103012085,
0.9050872921943665, 0.9056794047355652, 0.9041471481323242, 0.9061471223831177,
0.9084997773170471, 0.9083557724952698, 0.9094218611717224, 0.9110995531082153,
0.9115979075431824, 0.9128410220146179, 0.9144185781478882, 0.9135100841522217],
'val_loss': [2.8570048809051514, 2.732267379760742, 2.658287286758423,
2.689528703689575, 2.6541779041290283, 2.555762529373169, 2.5285749435424805,
2.5031163692474365, 2.4860129356384277, 2.4744083881378174, 2.5310981273651123,
2.4687845706939697, 2.6593470573425293, 2.327998399734497, 2.3269829750061035,
2.374300003051758, 2.3490216732025146, 2.686370849609375, 2.2187788486480713,
2.1920998096466064, 2.357571840286255, 2.468463182449341, 2.155719518661499,
2.2990543842315674, 2.3298680782318115, 2.179563522338867, 2.206254243850708,

2.1054115295410156, 2.408705949783325, 2.222367525100708, 2.0841636657714844,
2.3708086013793945, 2.151578664779663, 2.3537638187408447, 2.082036018371582,
2.0706701278686523, 2.267359733581543, 1.956411600112915, 2.0623412132263184,
2.063234567642212], 'val_accuracy': [0.2356899231672287, 0.2542169690132141,
0.26002395153045654, 0.25836482644081116, 0.26454052329063416, 0.2748640477657318,
0.2767075300216675, 0.289888471364975, 0.2912710905075073, 0.2822380065917969,
0.2888745367527008, 0.29873719811439514, 0.26942574977874756, 0.332104355096817,
0.3446400463581085, 0.3261130154132843, 0.32620519399642944, 0.23651950061321259,
0.37450456619262695, 0.36150798201560974, 0.3301686644554138, 0.31044337153434753,
0.37404370307922363, 0.33680522441864014, 0.33090606331825256, 0.3682366907596588,
0.36786800622940063, 0.3825237452983856, 0.3466678857803345, 0.36132362484931946,
0.40897777676582336, 0.34556180238723755, 0.3882385492324829, 0.3320121765136719,
0.38261592388153076, 0.3978246748447418, 0.3631671071052551, 0.43303531408309937,
0.41533783078193665, 0.39957600831985474], 'val_precision': [0.5597248077392578,
0.7219251394271851, 0.7050197124481201, 0.9327354431152344, 0.6486486196517944,
0.7125154733657837, 0.7113866806030273, 0.7494134306907654, 0.817829430103302,
0.7779605388641357, 0.8044490218162537, 0.6464232206344604, 0.9316860437393188,
0.7660818696022034, 0.8288016319274902, 0.71039879322052, 0.6660633683204651,
0.7183098793029785, 0.763851523399353, 0.755972683429718, 0.7325893044471741,
0.7911832928657532, 0.7891321778297424, 0.556302547454834, 0.6633266806602478,
0.7403621673583984, 0.7626414895057678, 0.7858695387840271, 0.654656171798706,
0.72185218334198, 0.7917364835739136, 0.6265466809272766, 0.7392283082008362,
0.8044464588165283, 0.7286630272865295, 0.7370916604995728, 0.7155619859695435,
0.8452102541923523, 0.7889009714126587, 0.813988983631134], 'val_recall':
[0.08249608427286148, 0.03733063116669655, 0.11521799117326736, 0.05751682072877884,
0.0884874165058136, 0.15900082886219025, 0.11632408201694489, 0.1472025066614151,
0.1361415833234787, 0.1307954639196396, 0.14333118498325348, 0.15992256999015808,
0.059083785861730576, 0.16904783248901367, 0.15171904861927032, 0.17568439245224,
0.20352105796337128, 0.07991520315408707, 0.13088764250278473, 0.20416627824306488,
0.151258185505867, 0.09429440647363663, 0.17937137186527252, 0.2135680764913559,
0.18305835127830505, 0.23366208374500275, 0.18628445267677307, 0.1999262571334839,
0.2378099411725998, 0.20978891849517822, 0.2614065706729889, 0.20536455512046814,
0.21190893650054932, 0.16342520713806152, 0.2722831666469574, 0.25790396332740784,
0.2288690209388733, 0.266752690076828, 0.26730573177337646, 0.24564476311206818],
'val_auc': [0.8075581192970276, 0.8382971882820129, 0.8522612452507019,
0.8492308855056763, 0.8572903275489807, 0.866297721862793, 0.8703784942626953,
0.8747201561927795, 0.8756064772605896, 0.8769789934158325, 0.8629456758499146,
0.8781158924102783, 0.8463805317878723, 0.8937369585037231, 0.8925958275794983,
0.8848066926002502, 0.8897116184234619, 0.8413214087486267, 0.9062564373016357,
0.9063124060630798, 0.8878251314163208, 0.8779076337814331, 0.9120509028434753,

0.8970903754234314, 0.8909077644348145, 0.9047569632530212, 0.902860701084137, 0.9178109765052795, 0.8762969374656677, 0.9042683839797974, 0.912990927696228, 0.8852710723876953, 0.9106762409210205, 0.8833408951759338, 0.9160290956497192, 0.9176627993583679, 0.894540548324585, 0.9279512166976929, 0.9150317907333374, 0.9199819564819336]}

NASNetLarge with 0.2 validation split after 40 epochs:

====> Statistics: epochs=40, batch_size=32, validation_split=0.2, lr=0.001, momentum=0.9, steps_per_epoch =100, feature shape= (256, 256, 3), no_classes=38, loss_function=categorical_crossentropy

Epoch 1/40

1358/1358 - 1833s - loss: 3.5177 - accuracy: 0.1115 - precision: 0.0928 - recall: 9.4348e-04 - auc: 0.7153 - val_loss: 3.1883 - val_accuracy: 0.1233 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7575

Epoch 2/40

1358/1358 - 1828s - loss: 3.1926 - accuracy: 0.1331 - precision: 0.6010 - recall: 0.0057 - auc: 0.7525 - val_loss: 3.1350 - val_accuracy: 0.1303 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7622

Epoch 3/40

1358/1358 - 1843s - loss: 3.1542 - accuracy: 0.1464 - precision: 0.6390 - recall: 0.0101 - auc: 0.7620 - val_loss: 3.1133 - val_accuracy: 0.1703 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7782

Epoch 4/40

1358/1358 - 2135s - loss: 3.1251 - accuracy: 0.1530 - precision: 0.7103 - recall: 0.0181 - auc: 0.7696 - val_loss: 3.0890 - val_accuracy: 0.1545 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7752

Epoch 5/40

1358/1358 - 1978s - loss: 3.1080 - accuracy: 0.1591 - precision: 0.7010 - recall: 0.0215 - auc: 0.7725 - val_loss: 3.0663 - val_accuracy: 0.1673 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7802

Epoch 6/40

1358/1358 - 1896s - loss: 3.0939 - accuracy: 0.1636 - precision: 0.7203 - recall: 0.0245 - auc: 0.7750 - val_loss: 3.0654 - val_accuracy: 0.1849 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7815

Epoch 7/40

1358/1358 - 1864s - loss: 3.0810 - accuracy: 0.1670 - precision: 0.7086 - recall: 0.0262 - auc: 0.7765 - val_loss: 3.0429 - val_accuracy: 0.1723 - val_precision: 0.9915 - val_recall: 0.0107 - val_auc: 0.7838

Epoch 8/40

```
1358/1358 - 1960s - loss: 3.0775 - accuracy: 0.1693 - precision: 0.6905 - recall:
0.0291 - auc: 0.7774 - val_loss: 3.0407 - val_accuracy: 0.1749 - val_precision: 0.8765
- val_recall: 0.0536 - val_auc: 0.7836
Epoch 9/40
1358/1358 - 1845s - loss: 3.0666 - accuracy: 0.1707 - precision: 0.7116 - recall:
0.0293 - auc: 0.7798 - val_loss: 3.0942 - val_accuracy: 0.1525 - val_precision: 0.7446
- val_recall: 0.0672 - val_auc: 0.7750
Epoch 10/40
1358/1358 - 1866s - loss: 3.0610 - accuracy: 0.1712 - precision: 0.7148 - recall:
0.0304 - auc: 0.7811 - val_loss: 3.0330 - val_accuracy: 0.1865 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7885
Epoch 11/40
1358/1358 - 1876s - loss: 3.0521 - accuracy: 0.1739 - precision: 0.7117 - recall:
0.0304 - auc: 0.7826 - val_loss: 3.0433 - val_accuracy: 0.1916 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7819
Epoch 12/40
1358/1358 - 1871s - loss: 3.0519 - accuracy: 0.1755 - precision: 0.6880 - recall:
0.0291 - auc: 0.7828 - val_loss: 3.0821 - val_accuracy: 0.1912 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7739
Epoch 13/40
1358/1358 - 1860s - loss: 3.0444 - accuracy: 0.1763 - precision: 0.6739 - recall:
0.0294 - auc: 0.7837 - val_loss: 3.0648 - val_accuracy: 0.1931 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7747
Epoch 14/40
1358/1358 - 1900s - loss: 3.0395 - accuracy: 0.1770 - precision: 0.6637 - recall:
0.0293 - auc: 0.7842 - val_loss: 3.0194 - val_accuracy: 0.1857 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7912
Epoch 15/40
1358/1358 - 1891s - loss: 3.0428 - accuracy: 0.1746 - precision: 0.6559 - recall:
0.0282 - auc: 0.7846 - val_loss: 3.0234 - val_accuracy: 0.1759 - val_precision: 0.7788
- val_recall: 0.0523 - val_auc: 0.7864
Epoch 16/40
1358/1358 - 1872s - loss: 3.0378 - accuracy: 0.1764 - precision: 0.6413 - recall:
0.0283 - auc: 0.7856 - val_loss: 3.0657 - val_accuracy: 0.1651 - val_precision: 0.6342
- val_recall: 0.0690 - val_auc: 0.7832
Epoch 17/40
1358/1358 - 1873s - loss: 3.0396 - accuracy: 0.1761 - precision: 0.6594 - recall:
0.0280 - auc: 0.7847 - val_loss: 3.0627 - val_accuracy: 0.1805 - val_precision: 0.8472
- val_recall: 0.0225 - val_auc: 0.7733
Epoch 18/40
```

```
1358/1358 - 1855s - loss: 3.0427 - accuracy: 0.1741 - precision: 0.6186 - recall:
0.0285 - auc: 0.7851 - val_loss: 3.1122 - val_accuracy: 0.1664 - val_precision: 0.3483
- val_recall: 0.0808 - val_auc: 0.7853
Epoch 19/40
1358/1358 - 1874s - loss: 3.0282 - accuracy: 0.1765 - precision: 0.6417 - recall:
0.0306 - auc: 0.7884 - val_loss: 2.9919 - val_accuracy: 0.1749 - val_precision: 0.7442
- val_recall: 0.0534 - val_auc: 0.7951
Epoch 20/40
1358/1358 - 1864s - loss: 3.0318 - accuracy: 0.1756 - precision: 0.6025 - recall:
0.0270 - auc: 0.7876 - val_loss: 3.1341 - val_accuracy: 0.1509 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7594
Epoch 21/40
1358/1358 - 1823s - loss: 3.0229 - accuracy: 0.1774 - precision: 0.6378 - recall:
0.0267 - auc: 0.7891 - val_loss: 3.1656 - val_accuracy: 0.1663 - val_precision: 0.3711
- val_recall: 0.0804 - val_auc: 0.7737
Epoch 22/40
1358/1358 - 1881s - loss: 3.0252 - accuracy: 0.1777 - precision: 0.6163 - recall:
0.0272 - auc: 0.7885 - val_loss: 3.0230 - val_accuracy: 0.1726 - val_precision: 0.6734
- val_recall: 0.0616 - val_auc: 0.7923
Epoch 23/40
1358/1358 - 1867s - loss: 3.0204 - accuracy: 0.1796 - precision: 0.6211 - recall:
0.0278 - auc: 0.7891 - val_loss: 2.9776 - val_accuracy: 0.1757 - val_precision: 0.7031
- val_recall: 0.0522 - val_auc: 0.7993
Epoch 24/40
1358/1358 - 1889s - loss: 3.0173 - accuracy: 0.1793 - precision: 0.6004 - recall:
0.0257 - auc: 0.7904 - val_loss: 2.9735 - val_accuracy: 0.1866 - val_precision: 0.7370
- val_recall: 0.0183 - val_auc: 0.7995
Epoch 25/40
1358/1358 - 1906s - loss: 3.0226 - accuracy: 0.1781 - precision: 0.5760 - recall:
0.0258 - auc: 0.7898 - val_loss: 3.0833 - val_accuracy: 0.1746 - val_precision: 0.7248
- val_recall: 0.0199 - val_auc: 0.7775
Epoch 26/40
1358/1358 - 1964s - loss: 3.0149 - accuracy: 0.1797 - precision: 0.6113 - recall:
0.0247 - auc: 0.7905 - val_loss: 3.0106 - val_accuracy: 0.1631 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7912
Epoch 27/40
1358/1358 - 1942s - loss: 3.0152 - accuracy: 0.1768 - precision: 0.5605 - recall:
0.0240 - auc: 0.7911 - val_loss: 3.1784 - val_accuracy: 0.1670 - val_precision: 0.3786
- val_recall: 0.0791 - val_auc: 0.7718
Epoch 28/40
```

```
1358/1358 - 1928s - loss: 3.0223 - accuracy: 0.1761 - precision: 0.5876 - recall:
0.0241 - auc: 0.7892 - val_loss: 2.9799 - val_accuracy: 0.1889 - val_precision: 0.5556
- val_recall: 9.2174e-04 - val_auc: 0.7984
Epoch 29/40
1358/1358 - 1833s - loss: 3.0148 - accuracy: 0.1759 - precision: 0.5538 - recall:
0.0228 - auc: 0.7913 - val_loss: 2.9888 - val_accuracy: 0.1934 - val_precision: 0.6835
- val_recall: 0.0263 - val_auc: 0.7995
Epoch 30/40
1358/1358 - 1854s - loss: 3.0197 - accuracy: 0.1790 - precision: 0.5676 - recall:
0.0226 - auc: 0.7900 - val_loss: 3.0372 - val_accuracy: 0.1818 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7839
Epoch 31/40
1358/1358 - 1869s - loss: 3.0112 - accuracy: 0.1794 - precision: 0.5683 - recall:
0.0231 - auc: 0.7919 - val_loss: 3.0027 - val_accuracy: 0.1760 - val_precision: 0.6042
- val_recall: 0.0604 - val_auc: 0.7952
Epoch 32/40
1358/1358 - 1896s - loss: 3.0029 - accuracy: 0.1793 - precision: 0.5601 - recall:
0.0229 - auc: 0.7933 - val_loss: 2.9843 - val_accuracy: 0.1754 - val_precision: 0.5763
- val_recall: 0.0619 - val_auc: 0.7994
Epoch 33/40
1358/1358 - 1871s - loss: 3.0067 - accuracy: 0.1769 - precision: 0.5375 - recall:
0.0228 - auc: 0.7929 - val_loss: 3.0618 - val_accuracy: 0.1871 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7749
Epoch 34/40
1358/1358 - 1837s - loss: 3.0044 - accuracy: 0.1780 - precision: 0.5422 - recall:
0.0204 - auc: 0.7935 - val_loss: 2.9847 - val_accuracy: 0.1764 - val_precision: 0.6011
- val_recall: 0.0583 - val_auc: 0.8010
Epoch 35/40
1358/1358 - 1816s - loss: 3.0023 - accuracy: 0.1787 - precision: 0.5293 - recall:
0.0206 - auc: 0.7938 - val_loss: 2.9637 - val_accuracy: 0.1827 - val_precision: 0.4870
- val_recall: 0.0052 - val_auc: 0.8020
Epoch 36/40
1358/1358 - 1813s - loss: 3.0060 - accuracy: 0.1774 - precision: 0.4983 - recall:
0.0203 - auc: 0.7935 - val_loss: 2.9712 - val_accuracy: 0.1908 - val_precision: 0.4068
- val_recall: 0.0044 - val_auc: 0.8012
Epoch 37/40
1358/1358 - 1790s - loss: 3.5031 - accuracy: 0.1242 - precision: 0.5436 - recall:
0.0066 - auc: 0.6487 - val_loss: 3.6786 - val_accuracy: 0.0938 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.5702
Epoch 38/40
```

```
1358/1358 - 1847s - loss: 3.5721 - accuracy: 0.0937 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.6162 - val_loss: 3.4852 - val_accuracy: 0.0938 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6459
Epoch 39/40
1358/1358 - 1783s - loss: 3.4358 - accuracy: 0.0937 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.6753 - val_loss: 3.3992 - val_accuracy: 0.0938 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6890
Epoch 40/40
1358/1358 - 1772s - loss: 3.3835 - accuracy: 0.0927 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.6909 - val_loss: 3.3715 - val_accuracy: 0.0987 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6920
{'loss': [3.517720937728882, 3.1926028728485107, 3.154221773147583,
3.1250967979431152, 3.1080267429351807, 3.0938873291015625, 3.0809810161590576,
3.077547550201416, 3.0666186809539795, 3.0610013008117676, 3.0520946979522705,
3.051879405975342, 3.0444180965423584, 3.039538860321045, 3.0428149700164795,
3.0378458499908447, 3.039632797241211, 3.042710304260254, 3.02816104888916,
3.031838893890381, 3.022930145263672, 3.025151252746582, 3.02043080329895,
3.017263412475586, 3.0225512981414795, 3.014864444732666, 3.015183925628662,
3.0223498344421387, 3.014805316925049, 3.019737720489502, 3.011244058609009,
3.0029451847076416, 3.006742477416992, 3.00437593460083, 3.0023043155670166,
3.006016254425049, 3.5031332969665527, 3.57211971282959, 3.4358036518096924,
3.3834826946258545], 'accuracy': [0.1115381047129631, 0.13310015201568604,
0.14637795090675354, 0.15298232436180115, 0.15912647545337677, 0.16363678872585297,
0.16697348654270172, 0.16932068765163422, 0.17074742913246155, 0.17120765149593353,
0.1738540083169937, 0.17553387582302094, 0.1763392835855484, 0.17698360979557037,
0.17459039390087128, 0.1763853132724762, 0.1761321723461151, 0.1740841269493103,
0.1765003651380539, 0.17562592029571533, 0.17744384706020355, 0.17774300277233124,
0.17960695922374725, 0.17926178872585297, 0.17813420295715332, 0.17972201108932495,
0.17675349116325378, 0.1760861575603485, 0.17594808340072632, 0.17903167009353638,
0.17939984798431396, 0.17928479611873627, 0.17686855792999268, 0.17799612879753113,
0.1787325143814087, 0.17735180258750916, 0.12417157739400864, 0.0937039777636528,
0.0937039777636528, 0.09273748099803925], 'precision': [0.09276018291711807,
0.6009732484817505, 0.6390101909637451, 0.7102888226509094, 0.7009767293930054,
0.7202702760696411, 0.7085927724838257, 0.6905412673950195, 0.711570680141449,
0.7148268222808838, 0.7117456793785095, 0.688043475151062, 0.6738786101341248,
0.663712203502655, 0.6559485793113708, 0.6412929892539978, 0.6594477295875549,
0.6185721158981323, 0.6417189836502075, 0.6024653315544128, 0.6377606987953186,
0.6163456439971924, 0.6211498975753784, 0.6004307866096497, 0.5759753584861755,
0.6112692356109619, 0.5604513883590698, 0.5876404643058777, 0.5537514090538025,
0.5676300525665283, 0.5683494210243225, 0.5600676536560059, 0.5374592542648315,
```

0.5421760678291321, 0.5293073058128357, 0.498301237821579, 0.5435606241226196, 0.0,
0.0, 0.0], 'recall': [0.0009434830863028765, 0.005683910101652145,
0.010102172382175922, 0.01811027154326439, 0.021469993516802788, 0.0245305597782135,
0.026187408715486526, 0.029063880443572998, 0.029293999075889587,
0.030398564413189888, 0.030398564413189888, 0.029132915660738945,
0.029386045411229134, 0.029293999075889587, 0.02816642075777054, 0.028304491192102432,
0.028028350323438644, 0.028511598706245422, 0.03058265894651413, 0.02699282020330429,
0.026739690452814102, 0.02724594995379448, 0.027844255790114403, 0.0256581362336874,
0.025819219648838043, 0.024714654311537743, 0.024001289159059525, 0.02407032437622547,
0.02275865152478218, 0.022597569972276688, 0.023057805374264717, 0.022850699722766876,
0.02278166450560093, 0.020411450415849686, 0.020572533831000328, 0.020250368863344193,
0.006604381371289492, 0.0, 0.0, 0.0], 'auc': [0.7153014540672302, 0.7524712681770325,
0.7620142102241516, 0.7695780396461487, 0.7725135087966919, 0.7750263214111328,
0.7764964699745178, 0.7773527503013611, 0.7797650098800659, 0.7811360955238342,
0.7825769186019897, 0.782802402973175, 0.7837092280387878, 0.7842039465904236,
0.7846376895904541, 0.7855895757675171, 0.7846812009811401, 0.7850984334945679,
0.7884116172790527, 0.7875784039497375, 0.7891284823417664, 0.7884814143180847,
0.7891452312469482, 0.7903957366943359, 0.7897759675979614, 0.7904508113861084,
0.791114866733551, 0.7891936898231506, 0.7912963628768921, 0.7899680137634277,
0.7918660044670105, 0.7933326959609985, 0.7929084897041321, 0.7934556603431702,
0.793825089931488, 0.7934644222259521, 0.6487067341804504, 0.6162338256835938,
0.6753333806991577, 0.6908961534500122], 'val_loss': [3.188253879547119,
3.1349825859069824, 3.113252878189087, 3.0889673233032227, 3.0663015842437744,
3.0653576850891113, 3.0429208278656006, 3.0407116413116455, 3.0942180156707764,
3.0329654216766357, 3.043333053588867, 3.082113742828369, 3.0647754669189453,
3.0193893909454346, 3.023432731628418, 3.0656630992889404, 3.062692880630493,
3.112204074859619, 2.991875410079956, 3.1340978145599365, 3.165602922439575,
3.022995710372925, 2.9776382446289062, 2.973531723022461, 3.083331346511841,
3.0105791091918945, 3.17844820022583, 2.979907989501953, 2.9887728691101074,
3.0372092723846436, 3.0027048587799072, 2.984271764755249, 3.0617787837982178,
2.9846832752227783, 2.963653802871704, 2.9712467193603516, 3.6786437034606934,
3.4852092266082764, 3.3991751670837402, 3.371462345123291], 'val_accuracy':
[0.12332934141159058, 0.13033458590507507, 0.17033827304840088, 0.15448428690433502,
0.16729652881622314, 0.18490183353424072, 0.17227394878864288, 0.1748548299074173,
0.15245644748210907, 0.18646879494190216, 0.19163057208061218, 0.19116969406604767,
0.19310535490512848, 0.1857314109802246, 0.17586874961853027, 0.1650843322277069,
0.18047747015953064, 0.1663747876882553, 0.17494699358940125, 0.15088948607444763,
0.16628260910511017, 0.17264263331890106, 0.17568439245224, 0.1865609735250473,
0.17457830905914307, 0.1631486713886261, 0.1670200079679489, 0.18886533379554749,
0.1933818757534027, 0.18176791071891785, 0.17596091330051422, 0.17540787160396576,

0.18711401522159576, 0.17642179131507874, 0.18268965184688568, 0.1908009946346283,
0.09383353590965271, 0.09383353590965271, 0.09383353590965271, 0.09871877729892731],
'val_precision': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9914529919624329,
0.8765060305595398, 0.7446373701095581, 0.0, 0.0, 0.0, 0.0, 0.0, 0.7788461446762085,
0.6342083215713501, 0.8472222089767456, 0.3482922911643982, 0.7442159652709961, 0.0,
0.37106382846832275, 0.6733871102333069, 0.7031055688858032, 0.7370370626449585,
0.7248322367668152, 0.0, 0.3786407709121704, 0.5555555820465088, 0.6834532618522644,
0.0, 0.6042435169219971, 0.5763293504714966, 0.0, 0.6011396050453186,
0.48695650696754456, 0.4067796468734741, 0.0, 0.0, 0.0, 0.0], 'val_recall': [0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.010692229494452477, 0.05364549905061722, 0.06719513237476349,
0.0, 0.0, 0.0, 0.0, 0.0, 0.05226287990808487, 0.06903862208127975,
0.022490551695227623, 0.0808369442820549, 0.05336897447705269, 0.0,
0.08037607371807098, 0.061572495847940445, 0.052170708775520325, 0.018342703580856323,
0.01990966871380806, 0.0, 0.07908563315868378, 0.0009217439219355583,
0.0262697022408247, 0.0, 0.06037422642111778, 0.06194119155406952, 0.0,
0.05834639072418213, 0.005161765962839127, 0.00442437082529068, 0.0, 0.0, 0.0, 0.0],
'val_auc': [0.7574712038040161, 0.7622308731079102, 0.7781664133071899,
0.775212287902832, 0.7801519632339478, 0.781473696231842, 0.7838215231895447,
0.7836127281188965, 0.7750425934791565, 0.7885153889656067, 0.7819359302520752,
0.7739196419715881, 0.7747398018836975, 0.7912003397941589, 0.7863665819168091,
0.7831682562828064, 0.7733379602432251, 0.7853420376777649, 0.7951160669326782,
0.7594369649887085, 0.7736616134643555, 0.7922553420066833, 0.7993282675743103,
0.7995116114616394, 0.7774758338928223, 0.7911561727523804, 0.7717729806900024,
0.7983630299568176, 0.7994722723960876, 0.7839487791061401, 0.7952304482460022,
0.7994034886360168, 0.7749067544937134, 0.8009639382362366, 0.8020088076591492,
0.8011947870254517, 0.5702069997787476, 0.645873486995697, 0.6890134811401367,
0.6920154094696045]}

MobileNet_v3_small with 0.2 validation split after 40 epochs:

====> Statistics: Model name: MobileNetV3Small_model, epochs=40, batch_size=32,
validation_split=0.2, lr=0.001, momentum=0.9, steps_per_epoch =100, feature shape=
(224, 224, 3), no_classes=38, loss_function=categorical_crossentropy
Epoch 1/40
1358/1358 - 3242s - loss: 3.3726 - accuracy: 0.0993 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.6949 - val_loss: 3.3496 - val_accuracy: 0.0987 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7031
Epoch 2/40
1358/1358 - 3271s - loss: 3.3506 - accuracy: 0.0989 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7018 - val_loss: 3.3538 - val_accuracy: 0.0987 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7083

```
Epoch 3/40
1358/1358 - 2666s - loss: 3.3370 - accuracy: 0.1077 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7066 - val_loss: 3.3169 - val_accuracy: 0.0984 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7191
Epoch 4/40
1358/1358 - 2486s - loss: 3.3011 - accuracy: 0.1217 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7190 - val_loss: 3.2432 - val_accuracy: 0.1283 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7384
Epoch 5/40
1358/1358 - 2339s - loss: 3.2477 - accuracy: 0.1361 - precision: 1.0000 - recall:
4.6024e-05 - auc: 0.7351 - val_loss: 3.3466 - val_accuracy: 0.1001 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7105
Epoch 6/40
1358/1358 - 2108s - loss: 3.1763 - accuracy: 0.1539 - precision: 0.5316 - recall:
0.0019 - auc: 0.7551 - val_loss: 3.0795 - val_accuracy: 0.1769 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7787
Epoch 7/40
1358/1358 - 2153s - loss: 3.1345 - accuracy: 0.1591 - precision: 0.5000 - recall:
0.0024 - auc: 0.7658 - val_loss: 3.0506 - val_accuracy: 0.1841 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7903
Epoch 8/40
1358/1358 - 2183s - loss: 3.1255 - accuracy: 0.1568 - precision: 0.4471 - recall:
0.0026 - auc: 0.7689 - val_loss: 3.0458 - val_accuracy: 0.1706 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7932
Epoch 9/40
1358/1358 - 2035s - loss: 3.0858 - accuracy: 0.1671 - precision: 0.5462 - recall:
0.0049 - auc: 0.7781 - val_loss: 3.0706 - val_accuracy: 0.1656 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7812
Epoch 10/40
1358/1358 - 1986s - loss: 3.0922 - accuracy: 0.1629 - precision: 0.5772 - recall:
0.0059 - auc: 0.7764 - val_loss: 3.0330 - val_accuracy: 0.1722 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7903
Epoch 11/40
1358/1358 - 2042s - loss: 3.0677 - accuracy: 0.1651 - precision: 0.5666 - recall:
0.0066 - auc: 0.7824 - val_loss: 3.1804 - val_accuracy: 0.1469 - val_precision: 0.6124
- val_recall: 0.0236 - val_auc: 0.7652
Epoch 12/40
1358/1358 - 2009s - loss: 3.0818 - accuracy: 0.1625 - precision: 0.5129 - recall:
0.0046 - auc: 0.7792 - val_loss: 3.0480 - val_accuracy: 0.1696 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7865
```

```
Epoch 13/40
1358/1358 - 1975s - loss: 3.0657 - accuracy: 0.1651 - precision: 0.5423 - recall:
0.0035 - auc: 0.7826 - val_loss: 3.0129 - val_accuracy: 0.1726 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7973
Epoch 14/40
1358/1358 - 1937s - loss: 3.0680 - accuracy: 0.1644 - precision: 0.5390 - recall:
0.0052 - auc: 0.7821 - val_loss: 2.9933 - val_accuracy: 0.1721 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7985
Epoch 15/40
1358/1358 - 1982s - loss: 3.0482 - accuracy: 0.1654 - precision: 0.5766 - recall:
0.0044 - auc: 0.7866 - val_loss: 2.9846 - val_accuracy: 0.1733 - val_precision: 0.3077
- val_recall: 3.6870e-04 - val_auc: 0.8028
Epoch 16/40
1358/1358 - 1950s - loss: 3.0521 - accuracy: 0.1659 - precision: 0.5558 - recall:
0.0067 - auc: 0.7855 - val_loss: 2.9897 - val_accuracy: 0.1702 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8032
Epoch 17/40
1358/1358 - 1975s - loss: 3.0409 - accuracy: 0.1655 - precision: 0.4935 - recall:
0.0044 - auc: 0.7888 - val_loss: 3.3546 - val_accuracy: 0.1432 - val_precision: 0.4879
- val_recall: 0.0539 - val_auc: 0.7431
Epoch 18/40
1358/1358 - 1962s - loss: 3.0394 - accuracy: 0.1668 - precision: 0.5293 - recall:
0.0052 - auc: 0.7890 - val_loss: 3.0612 - val_accuracy: 0.1616 - val_precision: 0.6012
- val_recall: 0.0277 - val_auc: 0.7864
Epoch 19/40
1358/1358 - 2025s - loss: 3.0404 - accuracy: 0.1649 - precision: 0.5331 - recall:
0.0044 - auc: 0.7886 - val_loss: 3.0058 - val_accuracy: 0.1701 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7958
Epoch 20/40
1358/1358 - 1934s - loss: 3.0354 - accuracy: 0.1643 - precision: 0.6031 - recall:
0.0054 - auc: 0.7895 - val_loss: 2.9987 - val_accuracy: 0.1645 - val_precision: 0.5840
- val_recall: 0.0128 - val_auc: 0.7986
Epoch 21/40
1358/1358 - 1894s - loss: 3.0284 - accuracy: 0.1666 - precision: 0.5668 - recall:
0.0073 - auc: 0.7911 - val_loss: 2.9835 - val_accuracy: 0.1717 - val_precision: 0.1667
- val_recall: 1.8435e-04 - val_auc: 0.8018
Epoch 22/40
1358/1358 - 1908s - loss: 3.0190 - accuracy: 0.1681 - precision: 0.5793 - recall:
0.0055 - auc: 0.7934 - val_loss: 2.9757 - val_accuracy: 0.1695 - val_precision: 0.5000
- val_recall: 1.8435e-04 - val_auc: 0.8033
```

```
Epoch 23/40
1358/1358 - 1919s - loss: 3.0266 - accuracy: 0.1670 - precision: 0.5184 - recall:
0.0058 - auc: 0.7914 - val_loss: 3.0197 - val_accuracy: 0.1631 - val_precision: 1.0000
- val_recall: 9.2174e-05 - val_auc: 0.7950
Epoch 24/40
1358/1358 - 1990s - loss: 3.0276 - accuracy: 0.1676 - precision: 0.5525 - recall:
0.0059 - auc: 0.7911 - val_loss: 3.2865 - val_accuracy: 0.1384 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7249
Epoch 25/40
1358/1358 - 2080s - loss: 3.0235 - accuracy: 0.1663 - precision: 0.5734 - recall:
0.0058 - auc: 0.7922 - val_loss: 2.9796 - val_accuracy: 0.1710 - val_precision: 0.2500
- val_recall: 5.5305e-04 - val_auc: 0.8022
Epoch 26/40
1358/1358 - 2009s - loss: 3.0188 - accuracy: 0.1685 - precision: 0.5706 - recall:
0.0064 - auc: 0.7932 - val_loss: 2.9892 - val_accuracy: 0.1655 - val_precision: 0.5786
- val_recall: 0.0170 - val_auc: 0.8013
Epoch 27/40
1358/1358 - 2051s - loss: 3.0231 - accuracy: 0.1683 - precision: 0.5660 - recall:
0.0058 - auc: 0.7922 - val_loss: 3.0009 - val_accuracy: 0.1741 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8027
Epoch 28/40
1358/1358 - 2296s - loss: 3.0170 - accuracy: 0.1692 - precision: 0.5228 - recall:
0.0047 - auc: 0.7937 - val_loss: 2.9807 - val_accuracy: 0.1670 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8021
Epoch 29/40
1358/1358 - 2510s - loss: 3.0148 - accuracy: 0.1687 - precision: 0.5685 - recall:
0.0057 - auc: 0.7940 - val_loss: 2.9942 - val_accuracy: 0.1655 - val_precision: 0.3421
- val_recall: 0.0012 - val_auc: 0.8006
Epoch 30/40
1358/1358 - 2763s - loss: 3.0161 - accuracy: 0.1691 - precision: 0.5847 - recall:
0.0051 - auc: 0.7934 - val_loss: 2.9693 - val_accuracy: 0.1722 - val_precision: 0.2500
- val_recall: 1.8435e-04 - val_auc: 0.8044
Epoch 31/40
1358/1358 - 2764s - loss: 3.0168 - accuracy: 0.1689 - precision: 0.5648 - recall:
0.0067 - auc: 0.7933 - val_loss: 3.0503 - val_accuracy: 0.1690 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7839
Epoch 32/40
1358/1358 - 2317s - loss: 3.0102 - accuracy: 0.1694 - precision: 0.5439 - recall:
0.0058 - auc: 0.7952 - val_loss: 3.0005 - val_accuracy: 0.1744 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8031
```

```
Epoch 33/40
1358/1358 - 2347s - loss: 3.0103 - accuracy: 0.1700 - precision: 0.5445 - recall:
0.0058 - auc: 0.7947 - val_loss: 2.9671 - val_accuracy: 0.1760 - val_precision: 1.0000
- val_recall: 9.2174e-05 - val_auc: 0.8034
Epoch 34/40
1358/1358 - 2278s - loss: 3.0065 - accuracy: 0.1695 - precision: 0.5743 - recall:
0.0053 - auc: 0.7956 - val_loss: 3.0769 - val_accuracy: 0.1589 - val_precision: 0.6043
- val_recall: 0.0312 - val_auc: 0.7841
Epoch 35/40
1358/1358 - 2223s - loss: 3.0061 - accuracy: 0.1711 - precision: 0.5972 - recall:
0.0058 - auc: 0.7960 - val_loss: 2.9628 - val_accuracy: 0.1708 - val_precision: 0.5985
- val_recall: 0.0146 - val_auc: 0.8055
Epoch 36/40
1358/1358 - 2272s - loss: 3.0036 - accuracy: 0.1715 - precision: 0.5737 - recall:
0.0057 - auc: 0.7961 - val_loss: 3.0271 - val_accuracy: 0.1862 - val_precision: 0.5435
- val_recall: 0.0069 - val_auc: 0.7948
Epoch 37/40
1358/1358 - 2268s - loss: 3.0041 - accuracy: 0.1725 - precision: 0.5567 - recall:
0.0063 - auc: 0.7963 - val_loss: 2.9692 - val_accuracy: 0.1696 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8039
Epoch 38/40
1358/1358 - 2217s - loss: 2.9927 - accuracy: 0.1739 - precision: 0.5223 - recall:
0.0057 - auc: 0.7983 - val_loss: 3.1058 - val_accuracy: 0.1591 - val_precision: 0.5524
- val_recall: 0.0126 - val_auc: 0.7765
Epoch 39/40
1358/1358 - 2263s - loss: 2.9961 - accuracy: 0.1743 - precision: 0.5881 - recall:
0.0078 - auc: 0.7975 - val_loss: 2.9568 - val_accuracy: 0.1746 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8079
Epoch 40/40
1358/1358 - 2252s - loss: 2.9898 - accuracy: 0.1737 - precision: 0.5501 - recall:
0.0072 - auc: 0.7990 - val_loss: 2.9603 - val_accuracy: 0.1724 - val_precision: 0.2414
- val_recall: 6.4522e-04 - val_auc: 0.8065
2021-01-13 17:11:27.238468: W tensorflow/python/util/util.cc:348] Sets are not
currently considered sequences, but this may change in the future, so consider
avoiding using them.
WARNING:tensorflow:AutoGraph could not transform <function
canonicalize_signatures.<locals>.signature_wrapper at 0x2b8d86f982f0> and will run it
as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to
10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
```

```
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
{'loss': [3.3725714683532715, 3.350614070892334, 3.3370277881622314,
3.3010849952697754, 3.247711420059204, 3.1763393878936768, 3.134514331817627,
3.125530242919922, 3.0857973098754883, 3.0921993255615234, 3.067735433578491,
3.0817816257476807, 3.0656931400299072, 3.067995071411133, 3.0482399463653564,
3.052074670791626, 3.0409135818481445, 3.0393900871276855, 3.040415048599243,
3.035382032394409, 3.0283939838409424, 3.019030809402466, 3.026578664779663,
3.0276365280151367, 3.0234999656677246, 3.018815279006958, 3.023099422454834,
3.016968011856079, 3.014803409576416, 3.016101360321045, 3.0168139934539795,
3.0102334022521973, 3.0103156566619873, 3.0064876079559326, 3.0061330795288086,
3.0036461353302, 3.0041275024414062, 2.992732048034668, 2.996108055114746,
2.989811420440674], 'accuracy': [0.09929583966732025, 0.09890463948249817,
0.10767212510108948, 0.12173232436180115, 0.1361377090215683, 0.1538797914981842,
0.15914948284626007, 0.15677927434444427, 0.16706553101539612, 0.1628543883562088,
0.1650865226984024, 0.16250920295715332, 0.1651325523853302, 0.16444219648838043,
0.1653626710176468, 0.16589193046092987, 0.1655007302761078, 0.16683541238307953,
0.16485640406608582, 0.16430412232875824, 0.16658228635787964, 0.16810107231140137,
0.16697348654270172, 0.1675948053598404, 0.16632916033267975, 0.16849227249622345,
0.16828516125679016, 0.16915960609912872, 0.16872239112854004, 0.16909056901931763,
0.16888347268104553, 0.169366717338562, 0.17003405094146729, 0.1694587618112564,
0.17111560702323914, 0.17148379981517792, 0.17254234850406647, 0.1738770306110382,
0.1743142455816269, 0.17369292676448822], 'precision': [0.0, 0.0, 0.0, 0.0, 1.0,
0.5316455960273743, 0.5, 0.4470588266849518, 0.5461538434028625, 0.5771812200546265,
0.5666003823280334, 0.5128865838050842, 0.5422534942626953, 0.5390070676803589,
0.5765765905380249, 0.5557692050933838, 0.49354004859924316, 0.5292739868164062,
0.5331491827964783, 0.6030927896499634, 0.5668449401855469, 0.5793269276618958,
0.5184426307678223, 0.5524625182151794, 0.5733634233474731, 0.5705521702766418,
0.5659955143928528, 0.5228426456451416, 0.568493127822876, 0.5846560597419739,
0.5647969245910645, 0.5438972115516663, 0.5444685220718384, 0.5742574334144592,
0.5971564054489136, 0.5737327337265015, 0.5566801428794861, 0.522292971611023,
0.5881326198577881, 0.5500878691673279], 'recall': [0.0, 0.0, 0.0, 0.0,
4.6023564209463075e-05, 0.0019329896895214915, 0.0024392488412559032,
0.0026233431417495012, 0.004901509732007980, 0.005937039852142334,
0.006558357737958431, 0.004579344764351845, 0.0035438144113868475,
0.0052466862834990025, 0.004418262280523777, 0.006650405004620552,
0.004395250231027603, 0.005200662650167942, 0.0044412738643586636,
0.005384757183492184, 0.00731774652376771, 0.0055458396673202515,
0.005821981001645327, 0.005937039852142334, 0.005844992585480213,
```

0.006420287303626537, 0.005821981001645327, 0.004740427248179913,
0.005729933734983206, 0.005085603799670935, 0.006719440221786499,
0.005844992585480213, 0.005775957368314266, 0.005338733550161123,
0.005798968952149153, 0.005729933734983206, 0.0063282400369644165,
0.005660898517817259, 0.007754970341920853, 0.007202687673270702], 'auc':
[0.6948797106742859, 0.7017914652824402, 0.7066015005111694, 0.7189710736274719,
0.7351499795913696, 0.755124568939209, 0.7658149600028992, 0.7688806056976318,
0.778132438659668, 0.7764497995376587, 0.7823501229286194, 0.7791545391082764,
0.7826332449913025, 0.782135546207428, 0.7866058349609375, 0.7855194807052612,
0.7887592911720276, 0.7889989614486694, 0.7885711789131165, 0.7894967198371887,
0.7911359071731567, 0.7934423685073853, 0.7914390563964844, 0.7911309003829956,
0.7922177910804749, 0.7931883335113525, 0.792178750038147, 0.7937297224998474,
0.7939899563789368, 0.7933627963066101, 0.7932713031768799, 0.7951580286026001,
0.7946604490280151, 0.7956212162971497, 0.7960137128829956, 0.796118438243866,
0.7962744235992432, 0.7982627749443054, 0.7974962592124939, 0.7990050315856934],
'val_loss': [3.3496196269989014, 3.353753089904785, 3.3169150352478027,
3.2432470321655273, 3.3465688228607178, 3.0794947147369385, 3.0505788326263428,
3.0458388328552246, 3.070613145828247, 3.0330214500427246, 3.1804497241973877,
3.047990083694458, 3.0128815174102783, 2.9933440685272217, 2.9846484661102295,
2.9896738529205322, 3.3546361923217773, 3.0612175464630127, 3.0057995319366455,
2.9987080097198486, 2.983541250228882, 2.975745916366577, 3.0196902751922607,
3.2865233421325684, 2.9795796871185303, 2.989222526550293, 3.000910520553589,
2.980706214904785, 2.9941954612731934, 2.969264507293701, 3.0503413677215576,
3.0004818439483643, 2.967109441757202, 3.0768933296203613, 2.9627814292907715,
3.0270919799804688, 2.96917462348938, 3.1058027744293213, 2.9568042755126953,
2.960294723510742], 'val_accuracy': [0.09871877729892731, 0.09871877729892731,
0.09844225645065308, 0.12830676138401031, 0.10010138899087906, 0.17688266932964325,
0.18407227098941803, 0.1706148087978363, 0.16563738882541656, 0.17218177020549774,
0.14692598581314087, 0.16960088908672333, 0.17255046963691711, 0.1720895916223526,
0.17328785359859467, 0.17024610936641693, 0.14323900640010834, 0.16158170998096466,
0.17006175220012665, 0.16453129053115845, 0.17172089219093323, 0.16950871050357819,
0.1631486713886261, 0.13844594359397888, 0.17098350822925568, 0.16545303165912628,
0.17411743104457855, 0.1670200079679489, 0.16554521024227142, 0.17218177020549774,
0.16904783248901367, 0.17439395189285278, 0.17596091330051422, 0.1589086502790451,
0.1707991510629654, 0.18619227409362793, 0.16960088908672333, 0.1590930074453354,
0.17457830905914307, 0.17236611247062683], 'val_precision': [0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.6124401688575745, 0.0, 0.0, 0.0, 0.3076923191547394, 0.0,
0.4879065752029419, 0.6012024283409119, 0.0, 0.5840336084365845, 0.1666666716337204,
0.5, 1.0, 0.0, 0.25, 0.5786163806915283, 0.0, 0.0, 0.34210526943206787, 0.25, 0.0,
0.0, 1.0, 0.6042780876159668, 0.5984848737716675, 0.54347825050354, 0.0,

0.5524193644523621, 0.0, 0.24137930572032928], 'val_recall': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.023596644401550293, 0.0, 0.0, 0.0, 0.00036869756877422333, 0.0, 0.05392201989889145, 0.02765231765806675, 0.0, 0.012812240980565548, 0.00018434878438711166, 0.00018434878438711166, 9.217439219355583e-05, 0.0, 0.000553046353161335, 0.016960088163614273, 0.0, 0.0, 0.0011982670985162258, 0.00018434878438711166, 0.0, 0.0, 9.217439219355583e-05, 0.03124712035059929, 0.014563553966581821, 0.006913079414516687, 0.0, 0.012627892196178436, 0.0, 0.0006452207453548908], 'val_auc': [0.7030975222587585, 0.7083134055137634, 0.7191049456596375, 0.7384495735168457, 0.7105422616004944, 0.7787346243858337, 0.7903229594230652, 0.7932428121566772, 0.7811580896377563, 0.7902934551239014, 0.7651557326316833, 0.786514163017273, 0.7973352670669556, 0.7985042929649353, 0.8027710914611816, 0.8031506538391113, 0.743131697177887, 0.7863640785217285, 0.7957621216773987, 0.7986395359039307, 0.8017964959144592, 0.803304135799408, 0.7950155735015869, 0.7248953580856323, 0.8022075295448303, 0.8013415932655334, 0.8027153015136719, 0.802111029624939, 0.800642728805542, 0.8043683767318726, 0.7838810682296753, 0.8030846118927002, 0.8034121990203857, 0.7840689420700073, 0.8054633140563965, 0.7948042750358582, 0.8039398193359375, 0.7764870524406433, 0.8079323172569275, 0.8064753413200378]}
date and time = 13-01-2021_17-11-21
Saving the model to path:
/users/PAA0023/dong760/plant_leaves_diagnosis/saved_models/MobileNetV3Small_model_BatchSize_32_0.2ValSplit_13-01-2021_17-11-21_40epochs

Prediction Result:
340/340 - 428s - loss: 2.9589 - accuracy: 0.1718 - precision: 0.2500 - recall: 5.5305e-04 - auc: 0.8067
Result: [2.9588820934295654, 0.17181307077407837, 0.25, 0.000553046353161335, 0.8067466020584106]
Validation accuracy: 0.17181307077407837 Validation accuracy: 2.9588820934295654

MobileNet_v3_large with 0.2 validation split after 40 epochs:

====> Statistics: Model name: MobileNetV3Large_model, epochs=40, batch_size=32, validation_split=0.2, lr=0.001, momentum=0.9, steps_per_epoch =100, feature shape=(256, 256, 3), no_classes=38, loss_function=categorical_crossentropy
Epoch 1/40
2021-01-12 18:32:40.315103: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library libcublas.so.11

```
2021-01-12 18:32:41.114954: I
tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened
dynamic library libcublasLt.so.11
2021-01-12 18:32:41.828243: I
tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened
dynamic library libcudnn.so.8
WARNING:tensorflow:AutoGraph could not transform <function
Model.make_test_function.<locals>.test_function at 0x2b5c28bd42f0> and will run it as-
is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to
10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
1358/1358 - 2733s - loss: 3.3634 - accuracy: 0.1030 - precision: 0.0462 - recall:
6.9035e-05 - auc: 0.6988 - val_loss: 3.3188 - val_accuracy: 0.1033 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7137
Epoch 2/40
1358/1358 - 2722s - loss: 3.2489 - accuracy: 0.1366 - precision: 0.2857 - recall:
4.6024e-05 - auc: 0.7371 - val_loss: 3.4194 - val_accuracy: 0.0938 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.6856
Epoch 3/40
1358/1358 - 2296s - loss: 3.1565 - accuracy: 0.1543 - precision: 1.0000 - recall:
2.3012e-05 - auc: 0.7633 - val_loss: 3.1042 - val_accuracy: 0.1601 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7783
Epoch 4/40
1358/1358 - 2284s - loss: 3.1279 - accuracy: 0.1593 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7700 - val_loss: 3.1083 - val_accuracy: 0.1591 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7758
Epoch 5/40
1358/1358 - 2376s - loss: 3.1170 - accuracy: 0.1604 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7721 - val_loss: 3.0791 - val_accuracy: 0.1625 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7823
Epoch 6/40
1358/1358 - 2224s - loss: 3.1026 - accuracy: 0.1632 - precision: 1.0000 - recall:
2.3012e-05 - auc: 0.7752 - val_loss: 3.0652 - val_accuracy: 0.1684 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7863
Epoch 7/40
```

```
1358/1358 - 2170s - loss: 3.1008 - accuracy: 0.1654 - precision: 0.8000 - recall:
9.2047e-05 - auc: 0.7754 - val_loss: 3.0986 - val_accuracy: 0.1474 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7772
Epoch 8/40
1358/1358 - 2222s - loss: 3.0886 - accuracy: 0.1636 - precision: 0.0000e+00 - recall:
0.0000e+00 - auc: 0.7782 - val_loss: 3.0485 - val_accuracy: 0.1631 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7876
Epoch 9/40
1358/1358 - 2185s - loss: 3.0846 - accuracy: 0.1679 - precision: 0.6250 - recall:
1.1506e-04 - auc: 0.7786 - val_loss: 3.0857 - val_accuracy: 0.1559 - val_precision:
1.0000 - val_recall: 9.2174e-05 - val_auc: 0.7844
Epoch 10/40
1358/1358 - 2135s - loss: 3.0789 - accuracy: 0.1661 - precision: 0.6667 - recall:
4.6024e-05 - auc: 0.7805 - val_loss: 3.0856 - val_accuracy: 0.1833 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7783
Epoch 11/40
1358/1358 - 2132s - loss: 3.0752 - accuracy: 0.1666 - precision: 0.4000 - recall:
1.3807e-04 - auc: 0.7807 - val_loss: 3.0454 - val_accuracy: 0.1686 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7902
Epoch 12/40
1358/1358 - 2115s - loss: 3.0759 - accuracy: 0.1696 - precision: 0.2000 - recall:
4.6024e-05 - auc: 0.7803 - val_loss: 3.0856 - val_accuracy: 0.1817 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7789
Epoch 13/40
1358/1358 - 2139s - loss: 3.0623 - accuracy: 0.1714 - precision: 0.6364 - recall:
1.6108e-04 - auc: 0.7832 - val_loss: 3.0658 - val_accuracy: 0.1946 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7912
Epoch 14/40
1358/1358 - 2151s - loss: 3.0528 - accuracy: 0.1715 - precision: 0.2727 - recall:
6.9035e-05 - auc: 0.7854 - val_loss: 3.0822 - val_accuracy: 0.1598 - val_precision:
0.2500 - val_recall: 1.8435e-04 - val_auc: 0.7800
Epoch 15/40
1358/1358 - 2148s - loss: 3.0557 - accuracy: 0.1740 - precision: 0.6111 - recall:
5.0626e-04 - auc: 0.7845 - val_loss: 3.0326 - val_accuracy: 0.1845 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7903
Epoch 16/40
1358/1358 - 2207s - loss: 3.0454 - accuracy: 0.1761 - precision: 0.4054 - recall:
3.4518e-04 - auc: 0.7867 - val_loss: 3.0425 - val_accuracy: 0.1727 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7862
Epoch 17/40
```

```
1358/1358 - 2096s - loss: 3.0390 - accuracy: 0.1790 - precision: 0.4918 - recall:
6.9035e-04 - auc: 0.7880 - val_loss: 3.0161 - val_accuracy: 0.1710 - val_precision:
0.1111 - val_recall: 9.2174e-05 - val_auc: 0.7927
Epoch 18/40
1358/1358 - 2074s - loss: 3.0354 - accuracy: 0.1813 - precision: 0.4960 - recall:
0.0014 - auc: 0.7886 - val_loss: 3.0588 - val_accuracy: 0.1718 - val_precision: 0.0714
- val_recall: 9.2174e-05 - val_auc: 0.7810
Epoch 19/40
1358/1358 - 2079s - loss: 3.0320 - accuracy: 0.1827 - precision: 0.5339 - recall:
0.0014 - auc: 0.7891 - val_loss: 3.0514 - val_accuracy: 0.2062 - val_precision: 0.6667
- val_recall: 1.8435e-04 - val_auc: 0.7938
Epoch 20/40
1358/1358 - 2135s - loss: 3.0348 - accuracy: 0.1836 - precision: 0.5889 - recall:
0.0024 - auc: 0.7891 - val_loss: 3.0247 - val_accuracy: 0.2012 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7914
Epoch 21/40
1358/1358 - 2224s - loss: 3.0184 - accuracy: 0.1860 - precision: 0.5230 - recall:
0.0021 - auc: 0.7920 - val_loss: 3.0683 - val_accuracy: 0.1710 - val_precision: 0.3667
- val_recall: 0.0010 - val_auc: 0.7804
Epoch 22/40
1358/1358 - 2170s - loss: 3.0189 - accuracy: 0.1854 - precision: 0.5603 - recall:
0.0030 - auc: 0.7923 - val_loss: 3.0619 - val_accuracy: 0.1774 - val_precision: 0.9286
- val_recall: 0.0024 - val_auc: 0.7811
Epoch 23/40
1358/1358 - 2185s - loss: 3.0167 - accuracy: 0.1900 - precision: 0.6128 - recall:
0.0038 - auc: 0.7917 - val_loss: 2.9754 - val_accuracy: 0.1796 - val_precision: 0.7874
- val_recall: 0.0126 - val_auc: 0.8004
Epoch 24/40
1358/1358 - 2619s - loss: 3.0048 - accuracy: 0.1910 - precision: 0.6164 - recall:
0.0043 - auc: 0.7947 - val_loss: 2.9690 - val_accuracy: 0.2066 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8027
Epoch 25/40
1358/1358 - 2838s - loss: 3.0061 - accuracy: 0.1908 - precision: 0.6528 - recall:
0.0051 - auc: 0.7937 - val_loss: 2.9561 - val_accuracy: 0.2114 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.8063
Epoch 26/40
1358/1358 - 3186s - loss: 2.9944 - accuracy: 0.1949 - precision: 0.5897 - recall:
0.0058 - auc: 0.7958 - val_loss: 2.9525 - val_accuracy: 0.2075 - val_precision: 0.3284
- val_recall: 0.0020 - val_auc: 0.8055
Epoch 27/40
```

```
1358/1358 - 2563s - loss: 2.9854 - accuracy: 0.1966 - precision: 0.6190 - recall:
0.0063 - auc: 0.7979 - val_loss: 2.9608 - val_accuracy: 0.1970 - val_precision: 0.8148
- val_recall: 0.0020 - val_auc: 0.8017
Epoch 28/40
1358/1358 - 2556s - loss: 2.9867 - accuracy: 0.1965 - precision: 0.5906 - recall:
0.0064 - auc: 0.7977 - val_loss: 2.9408 - val_accuracy: 0.2089 - val_precision: 0.4545
- val_recall: 4.6087e-04 - val_auc: 0.8074
Epoch 29/40
1358/1358 - 2523s - loss: 2.9856 - accuracy: 0.1967 - precision: 0.5961 - recall:
0.0078 - auc: 0.7979 - val_loss: 2.9421 - val_accuracy: 0.2028 - val_precision: 0.4964
- val_recall: 0.0064 - val_auc: 0.8057
Epoch 30/40
1358/1358 - 2415s - loss: 2.9835 - accuracy: 0.1987 - precision: 0.5608 - recall:
0.0073 - auc: 0.7984 - val_loss: 2.9369 - val_accuracy: 0.2146 - val_precision: 0.4118
- val_recall: 0.0019 - val_auc: 0.8077
Epoch 31/40
1358/1358 - 2447s - loss: 2.9755 - accuracy: 0.2001 - precision: 0.6372 - recall:
0.0093 - auc: 0.7996 - val_loss: 2.9476 - val_accuracy: 0.2076 - val_precision: 0.7692
- val_recall: 0.0028 - val_auc: 0.8035
Epoch 32/40
1358/1358 - 2446s - loss: 2.9689 - accuracy: 0.2036 - precision: 0.5828 - recall:
0.0084 - auc: 0.8009 - val_loss: 3.0612 - val_accuracy: 0.1809 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7869
Epoch 33/40
1358/1358 - 2447s - loss: 2.9786 - accuracy: 0.2014 - precision: 0.5978 - recall:
0.0102 - auc: 0.7989 - val_loss: 2.9246 - val_accuracy: 0.2131 - val_precision: 0.5000
- val_recall: 8.2957e-04 - val_auc: 0.8092
Epoch 34/40
1358/1358 - 2485s - loss: 2.9642 - accuracy: 0.2043 - precision: 0.6036 - recall:
0.0108 - auc: 0.8014 - val_loss: 2.9244 - val_accuracy: 0.2217 - val_precision: 0.2857
- val_recall: 1.8435e-04 - val_auc: 0.8119
Epoch 35/40
1358/1358 - 2510s - loss: 2.9680 - accuracy: 0.2032 - precision: 0.6390 - recall:
0.0119 - auc: 0.8007 - val_loss: 3.0154 - val_accuracy: 0.1990 - val_precision:
0.0000e+00 - val_recall: 0.0000e+00 - val_auc: 0.7925
Epoch 36/40
1358/1358 - 2435s - loss: 2.9565 - accuracy: 0.2052 - precision: 0.6469 - recall:
0.0128 - auc: 0.8029 - val_loss: 2.9733 - val_accuracy: 0.2079 - val_precision: 0.5556
- val_recall: 0.0023 - val_auc: 0.7971
Epoch 37/40
```

```
1358/1358 - 2435s - loss: 2.9658 - accuracy: 0.2028 - precision: 0.5816 - recall:
0.0131 - auc: 0.8011 - val_loss: 2.9617 - val_accuracy: 0.2176 - val_precision: 1.0000
- val_recall: 2.7652e-04 - val_auc: 0.8058
Epoch 38/40
1358/1358 - 2416s - loss: 2.9583 - accuracy: 0.2039 - precision: 0.5834 - recall:
0.0130 - auc: 0.8032 - val_loss: 2.9402 - val_accuracy: 0.2211 - val_precision: 0.4857
- val_recall: 0.0016 - val_auc: 0.8094
Epoch 39/40
1358/1358 - 2368s - loss: 2.9469 - accuracy: 0.2072 - precision: 0.5920 - recall:
0.0140 - auc: 0.8046 - val_loss: 2.9064 - val_accuracy: 0.2183 - val_precision: 0.5490
- val_recall: 0.0103 - val_auc: 0.8116
Epoch 40/40
1358/1358 - 2382s - loss: 2.9499 - accuracy: 0.2078 - precision: 0.6278 - recall:
0.0151 - auc: 0.8039 - val_loss: 2.9072 - val_accuracy: 0.2215 - val_precision: 0.4079
- val_recall: 0.0029 - val_auc: 0.8142
2021-01-13 20:43:59.444316: W tensorflow/python/util/util.cc:348] Sets are not
currently considered sequences, but this may change in the future, so consider
avoiding using them.
WARNING:tensorflow:AutoGraph could not transform <function
canonicalize_signatures.<locals>.signature_wrapper at 0x2b5c761017b8> and will run it
as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to
10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert
{'loss': [3.3634395599365234, 3.2489256858825684, 3.15647554397583, 3.127873659133911,
3.117034673690796, 3.102632522583008, 3.1008400917053223, 3.0885820388793945,
3.084585666656494, 3.0789124965667725, 3.0751683712005615, 3.07588529586792,
3.062344789505005, 3.0528128147125244, 3.0557000637054443, 3.0454037189483643,
3.0390496253967285, 3.0353991985321045, 3.0320353507995605, 3.034792423248291,
3.0183615684509277, 3.018899917602539, 3.016723155975342, 3.004828453063965,
3.0061280727386475, 2.99444580078125, 2.985361337661743, 2.9867138862609863,
2.9856393337249756, 2.9834775924682617, 2.97548508644104, 2.9688563346862793,
2.978597402572632, 2.9641802310943604, 2.9679617881774902, 2.9564850330352783,
2.965832233428955, 2.9582712650299072, 2.94691801071167, 2.949944257736206],
'accuracy': [0.10297772288322449, 0.13655191659927368, 0.1542709916830063,
0.15931056439876556, 0.16041512787342072, 0.1631765514612198, 0.1653856784105301,
0.16359075903892517, 0.16791696846485138, 0.16612204909324646, 0.16662831604480743,
0.1696198433637619, 0.17141476273536682, 0.1715298295021057, 0.1740381121635437,
```

0.1761091649532318, 0.17900864779949188, 0.18133284151554108, 0.1827135533094406,
0.18356499075889587, 0.18602724373340607, 0.1854059249162674, 0.1899622678756714,
0.19104380905628204, 0.19081369042396545, 0.19486376643180847, 0.196566641330719,
0.1964745968580246, 0.1967047154903412, 0.1986607164144516, 0.20006443560123444,
0.20356222987174988, 0.20137611031532288, 0.20429860055446625, 0.2032170444726944,
0.2052420824766159, 0.20284885168075562, 0.20386137068271637, 0.20717507600784302,
0.2078424096107483], 'precision': [0.04615384712815285, 0.2857142984867096, 1.0, 0.0,
0.0, 1.0, 0.800000011920929, 0.0, 0.625, 0.6666666865348816, 0.4000000059604645,
0.20000000298023224, 0.6363636255264282, 0.27272728085517883, 0.6111111044883728,
0.4054054021835327, 0.49180328845977783, 0.4959999918937683, 0.5338982939720154,
0.5888888835906982, 0.522988498210907, 0.5603448152542114, 0.6127819418907166,
0.6163934469223022, 0.6528189778327942, 0.5897436141967773, 0.6190476417541504,
0.5906183123588562, 0.5961199402809143, 0.5608465671539307, 0.6372239589691162,
0.5828025341033936, 0.5978407263755798, 0.6036036014556885, 0.6389577984809875,
0.6468531489372253, 0.581632673740387, 0.583419680595398, 0.5920155644416809,
0.6277511715888977], 'recall': [6.903534813318402e-05, 4.6023564209463075e-05,
2.3011782104731537e-05, 0.0, 0.0, 2.3011782104731537e-05, 9.204712841892615e-05, 0.0,
0.00011505890870466828, 4.6023564209463075e-05, 0.00013807069626636803,
4.6023564209463075e-05, 0.00016108246927615255, 6.903534813318402e-05,
0.00050625920994207 26, 0.00034517672611400485, 0.0006903534522280097,
0.0014267305377870798, 0.0014497422380372882, 0.0024392488412559032,
0.0020940720569342375, 0.002991531742736697, 0.0037509205285459757,
0.004326215013861656, 0.005062592215836048, 0.005821981001645327,
0.006282216403633356, 0.006374263670295477, 0.0077779823914170265,
0.00731774652376771, 0.009296759963035583, 0.008422312326729298, 0.010194219648838043,
0.010792525485157967, 0.011851067654788494, 0.012771539390087128,
0.013116715475916862, 0.012955632992088795, 0.013991163112223148,
0.015095729380846024], 'auc': [0.6988391280174255, 0.7370836138725281,
0.7632984519004822, 0.76997309923172, 0.7720807790756226, 0.7752304077148438,
0.7753975987434387, 0.7781652808189392, 0.7786258459091187, 0.7805317044258118,
0.7807421684265137, 0.7802738547325134, 0.7831559777259827, 0.7853884696960449,
0.7844577431678772, 0.7866986393928528, 0.7879537343978882, 0.7885648012161255,
0.7890802621841431, 0.7891010046005249, 0.791968584060669, 0.7923040986061096,
0.7916755676269531, 0.7946842908859253, 0.7936719059944153, 0.7958359718322754,
0.79793381690979, 0.7976589798927307, 0.797935426235199, 0.7983601093292236,
0.7996153831481934, 0.8008984327316284, 0.7988961338996887, 0.8013865351676941,
0.800690770149231, 0.8029046654701233, 0.8011335730552673, 0.8032068610191345,
0.8046476244926453, 0.8038834929466248], 'val_loss': [3.3188297748565674,
3.4193766117095947, 3.1041510105133057, 3.1083154678344727, 3.0790648460388184,
3.065214157104492, 3.0985567569732666, 3.0485446453094482, 3.085710048675537,

3.0856411457061768, 3.0453577041625977, 3.0856361389160156, 3.065824031829834,
3.0821597576141357, 3.032637596130371, 3.0425403118133545, 3.016054630279541,
3.0588324069976807, 3.051356554031372, 3.0246570110321045, 3.0682876110076904,
3.0619380474090576, 2.9753804206848145, 2.9689509868621826, 2.9561142921447754,
2.9525339603424072, 2.960834503173828, 2.940837860107422, 2.9420883655548096,
2.9368600845336914, 2.947605609893799, 3.061150074005127, 2.9246249198913574,
2.924431324005127, 3.015378713607788, 2.9733357429504395, 2.9617486000061035,
2.9401752948760986, 2.9063732624053955, 2.907215118408203], 'val_accuracy':
[0.10332749783992767, 0.09383353590965271, 0.16010692715644836, 0.1590930074453354,
0.1625034511089325, 0.16840261220932007, 0.1473868489265442, 0.1631486713886261,
0.15586690604686737, 0.18333487212657928, 0.16858696937561035, 0.1816757321357727,
0.19458015263080597, 0.15983040630817413, 0.18453313410282135, 0.1727348119020462,
0.17098350822925568, 0.17181307077407837, 0.20619411766529083, 0.2012166976928711,
0.17098350822925568, 0.1774357110261917, 0.1795557141304016, 0.2065628170967102,
0.2114480584859848, 0.20748455822467804, 0.1969766765832901, 0.2088671773672104,
0.20278367400169373, 0.21458199620246887, 0.20757673680782318, 0.18093833327293396,
0.21310719847679138, 0.2216794192790985, 0.19900451600551605, 0.2078532576560974,
0.2176237404346466, 0.22112637758255005, 0.2182689607143402, 0.22149506211280823],
'val_precision': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
0.25, 0.0, 0.0, 0.1111111119389534, 0.0714285746216774, 0.6666666865348816, 0.0,
0.36666667461395264, 0.9285714030265808, 0.7873563170433044, 0.0, 0.0,
0.3283582031726837, 0.8148148059844971, 0.4545454680919647, 0.49640288949012756,
0.4117647111415863, 0.7692307829856873, 0.0, 0.5, 0.2857142984867096, 0.0,
0.5555555820465088, 1.0, 0.48571428656578064, 0.5490196347236633,
0.40789473056793213], 'val_recall': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
9.217439219355583e-05, 0.0, 0.0, 0.0, 0.0, 0.00018434878438711166, 0.0, 0.0,
9.217439219355583e-05, 9.217439219355583e-05, 0.00018434878438711166, 0.0,
0.0010139183141291142, 0.0023965341970324516, 0.012627892196178436, 0.0, 0.0,
0.0020278366282582283, 0.0020278366282582283, 0.0004608719609677791 6,
0.006360033061355352, 0.0019356622360646725, 0.002765231765806675, 0.0,
0.0008295695297420025, 0.00018434878438711166, 0.0, 0.002304359804838896,
0.0002765231765806675, 0.0015669646672904491, 0.010323531925678253,
0.002857406158000231], 'val_auc': [0.713679850101471, 0.685649037361145,
0.778331995010376, 0.7758091688156128, 0.7823119163513184, 0.786266565322876,
0.7772420644760132, 0.787632942199707, 0.7843757271766663, 0.778266191482544,
0.7902413010597229, 0.77885502576828, 0.7912419438362122, 0.779979407787323,
0.7903074026107788, 0.7861600518226624, 0.7926756739616394, 0.7810245752334595,
0.7938271164894104, 0.7914190888404846, 0.7803940176963806, 0.7811294794082642,
0.8003998398780823, 0.8026575446128845, 0.8062604069709778, 0.8055236339569092,
0.8016547560691833, 0.8074143528938293, 0.8057374954223633, 0.8076844215393066,

```
0.8034720420837402, 0.7869330048561096, 0.8092291951179504, 0.8118849396705627,
0.7925330996513367, 0.7970616817474365, 0.8057863116264343, 0.8093887567520142,
0.8115602731704712, 0.8141567707061768]}
date and time = 13-01-2021_20-43-53
Saving the model to path:
/users/PAA0023/dong760/plant_leaves_diagnosis/saved_models/MobileNetV3Large_model_Batc
hSize_32_0.2ValSplit_13-01-2021_20-43-53_40epochs

Prediction Result:
340/340 - 456s - loss: 2.9051 - accuracy: 0.2243 - precision: 0.4896 - recall: 0.0043
- auc: 0.8142
Result: [2.905118227005005, 0.22426030039787292, 0.4895833432674408,
0.004332196433097124, 0.8141753077507019]
Validation accuracy: 0.22426030039787292 Validation accuracy: 2.905118227005005
```