```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
import os
```

```
data_dir = '/content/drive/MyDrive/potato_dataset'
categories = os.listdir(data_dir)
```

```
categories=sorted(categories)
print(categories)
```

```
['Potato___Early_blight.zip', 'Potato___Late_blight.zip', 'Potato___healthy.zip']
```

```
base_dir = '/content/drive/MyDrive/potato_dataset'
```

```
import zipfile

def unzip_file(zip_path, extract_to):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)

# Define paths
base_dir = '/content/drive/MyDrive/potato_dataset'
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')

# Unzip files
zip_files = [
    'Potato___Early_blight.zip',
    'Potato___healthy.zip',
    'Potato___Late_blight.zip'
]

# Unzip the main dataset files
for zip_file in zip_files:
    unzip_file(os.path.join(base_dir, zip_file), base_dir)

# Unzip the split dataset files if they exist
for zip_file in zip_files:
    train_zip = os.path.join(train_dir, zip_file)
    val_zip = os.path.join(val_dir, zip_file)
    if os.path.exists(train_zip):
        unzip_file(train_zip, train_dir)
    if os.path.exists(val_zip):
        unzip_file(val_zip, val_dir)
```

```
import shutil
import random
def organize_dataset(base_dir, train_dir, val_dir):
    classes = ['Potato___Early_blight', 'Potato___healthy', 'Potato___Late_blight']

    for cls in classes:
        cls_dir = os.path.join(base_dir, cls)
        if os.path.exists(cls_dir):
            images = os.listdir(cls_dir)
            random.shuffle(images)

            val_count = int(len(images) * 0.2)
            train_images = images[val_count:]
            val_images = images[:val_count]

            # Move images to train directory
            train_cls_dir = os.path.join(train_dir, cls)
            os.makedirs(train_cls_dir, exist_ok=True)
            for img in train_images:
                shutil.move(os.path.join(cls_dir, img), os.path.join(train_cls_dir, img))

            # Move images to validation directory
            val_cls_dir = os.path.join(val_dir, cls)
            os.makedirs(val_cls_dir, exist_ok=True)
            for img in val_images:
                shutil.move(os.path.join(cls_dir, img), os.path.join(val_cls_dir, img))

# Organize the dataset
organize_dataset(base_dir, train_dir, val_dir)



def verify_dataset_structure(directory):
    for dirpath, dirnames, filenames in os.walk(directory):
        print(f'Found directory: {dirpath}')
        for file_name in filenames:
            print(f'\t{file_name}')

print("Train directory structure:")
verify_dataset_structure(train_dir)

print("Validation directory structure:")
verify_dataset_structure(val_dir)
```

```
Found directory: /content/drive/MyDrive/potato_dataset/train/Potato___Early_blight
        e00d7014-a909-4d98-892b-da96a2be9a2e___RS_Early.B 7535.JPG
        c8dedd98-c5d7-4ef4-b4f4-70d0bae0e178___RS_Early.B 7128.JPG
        d825093a-2bd7-458d-a9a1-036db6c08dec___RS_Early.B 9018.JPG
        dc34096f-8974-4a76-8d15-42ee86038015___RS_Early.B 6734.JPG
        fc87399a-b45c-4b0f-a6c9-54f0e4f9d3c5___RS_Early.B 7277.JPG
        d998c614-017c-4f90-8e39-f8c500d21218___RS_Early.B 8292.JPG
        d8a55ed1-7b3a-4b25-84b2-6046c0d9e3ab___RS_Early.B 7443.JPG
        f56f07b9-9f3a-4c80-a4f8-daf49f479db2___RS_Early.B 8741.JPG
        f536e055-666e-41cc-8ee2-c1af2fbf754a___RS_Early.B 8104.JPG
        f164ce92-d109-47ca-9f75-380a7f16155a___RS_Early.B 7104.JPG
        f2540cea-220b-4ba2-bf16-7c1c7f32c38c___RS_Early.B 7997.JPG
        d6ffc732-27cc-4391-9820-be9d386fb245___RS_Early.B 7032.JPG
        cb5241ed-1a40-488a-9236-64ed07e6ebbf___RS_Early.B 6712.JPG
        e0042931-92f3-4527-b292-872075f4261d___RS_Early.B 7203.JPG
        cff1ed1b-51ec-4d44-ab1f-7a3dc1ec9ea9___RS_Early.B 7100.JPG
        f7a5e3e3-796e-4f4a-943c-24d26e2591d4___RS_Early.B 8679.JPG
        ed64bc0c-d2e5-44fd-a21a-f4d6b4f2f219___RS_Early.B 7366.JPG
        f188a6c1-fbed-4941-a5a7-e11a6b4ddcfb___RS_Early.B 6910.JPG
        e304fbf5-d145-433c-8f67-7b486581166a___RS_Early.B 7434.JPG
        e6d543a1-8e5e-4709-9329-b93d63f52edf___RS_Early.B 8324.JPG
        edc5f476-dd30-4fab-a8a1-71faf6210420___RS_Early.B 7315.JPG
        f15637d3-829d-46f9-b45e-1e1768d6b8c9___RS_Early.B 7635.JPG
        d1072068-911c-4656-8bf2-e662982199e1___RS_Early.B 8141.JPG
        d031102c-679d-4323-a5ee-4c52ed2d5740___RS_Early.B 6947.JPG
        caef0735-6517-4194-b124-5916815e4a71___RS_Early.B 7956.JPG
        f2a56e13-438e-4bef-a3ea-beb9f347d481___RS_Early.B 6691.JPG
        dcc9e3a4-04dc-46fd-9bc9-739334aa23d9___RS_Early.B 6799.JPG
        fc603fb2-e2a0-4990-8a75-0e4f80f40694___RS_Early.B 7584.JPG
        f5ec7cce-c3cc-4b4a-b716-d70175b1dcd2___RS_Early.B 7520.JPG
        e7cab6f5-308b-41ea-af82-e20cfe540729___RS_Early.B 7177.JPG
        d0be9a48-d0f6-4d34-9c80-3eb7631e2d8d___RS_Early.B 8143.JPG
        f838c192-a78c-460c-ab90-313dd5014a47___RS_Early.B 7718.JPG
        e016d105-f5cd-4082-a8c9-1913ea3fcfbb___RS_Early.B 7800.JPG
        d87f93fa-f6f8-49f1-b9e1-8f196f377df8___RS_Early.B 7244.JPG
        fdc1f5ed-66b5-4564-8957-055905b8a569___RS_Early.B 8244.JPG
        f0b444a4-ddd7-4286-8a80-423f7e71c526___RS_Early.B 8753.JPG
        fdc691b0-2b15-4cb6-8f5d-c4e5654389e0___RS_Early.B 7935.JPG
        ca1bc7af-3220-4b45-b76d-8f950128c489___RS_Early.B 7266.JPG
        d9b34eaa-9d54-41fe-9ea2-335fe0b572ee___RS_Early.B 7735.JPG
```

```
f686133a-e89a-4242-a52d-02f32ffd5275___RS_Early.B 8295.JPG
d286d101-227f-48ba-b906-586879eb6a00___RS_Early.B 7095.JPG
e11d63b8-93fa-41ba-b826-99811ee4c232___RS_Early.B 7387.JPG
e0654927-6cc0-491f-86d3-acb5ad261904___RS_Early.B 8341.JPG
ec6b34ed-69c6-466a-81a9-4ca18eb25275___RS_Early.B 7038.JPG
fa61b2e4-413c-4503-a2a2-cff2d8a11351___RS_Early.B 7370.JPG
db79941d-3ca1-42f1-b06e-d150a49d476a___RS_Early.B 8742.JPG
e786a4fe-5aa1-4da2-a16f-4ee82c56e317___RS_Early.B 7245.JPG
f9580dc3-d5d9-4990-a64d-3974a9d1c687___RS_Early.B 8755.JPG
e1710bad-79c1-4b36-bbd2-b257c50697a5___RS_Early.B 7426.JPG
fd59ab68-681f-4aca-ae95-6f73bf8caad7___RS_Early.B 7118.JPG
ccec0636-2984-4164-aad7-34b1b84ec42b___RS_Early.B 7766.JPG
e6d7262c-803d-4346-a36d-4f384196e21c___RS_Early.B 6893.JPG
de8d212e-1bbf-41d7-b13b-29f0746223aa___RS_Early.B 8022.JPG
ed270d5d-3523-4bc2-b208-4f5304bbfeef___RS_Early.B 8218.JPG
e3007682-ad4b-4a62-b64d-ebb0c4537077___RS_Early.B 7772.JPG
```

```python
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam


train_dir = '/content/drive/MyDrive/potato_dataset/train'
val_dir = '/content/drive/MyDrive/potato_dataset/validation'


img_height = 150
img_width = 150
batch_size = 32
```

ACO

```python
# Define the CNN model
def create_model(params):
    model = Sequential([
        Conv2D(params['filters1'], (params['kernel_size1'], params['kernel_size1']), activation='relu', input_shape=(img_hei
        MaxPooling2D((2, 2)),
        Conv2D(params['filters2'], (params['kernel_size2'], params['kernel_size2']), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(params['dense_units'], activation='relu'),
        Dropout(params['dropout']),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=params['learning_rate']),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model



# Define ACO parameters
class Ant:
    def __init__(self):
        self.position = None
        self.cost = float('inf')


def random_params():
    return {
        'filters1': np.random.choice([32, 64, 128]),
        'kernel_size1': np.random.choice([3, 5]),
        'filters2': np.random.choice([32, 64, 128]),
        'kernel_size2': np.random.choice([3, 5]),
        'dense_units': np.random.choice([128, 256, 512]),
        'dropout': np.random.uniform(0.2, 0.5),
        'learning_rate': np.random.choice([1e-3, 1e-4, 1e-5])
    }
```

```
def aco_optimize(num_ants, num_generations):
    best_ant = Ant()

    for generation in range(num_generations):
        ants = [Ant() for _ in range(num_ants)]

        for ant in ants:
            ant.position = random_params()
            model = create_model(ant.position)
            history = model.fit(train_generator, epochs=5, validation_data=validation_generator, verbose=0)
            val_accuracy = history.history['val_accuracy'][-1]
            ant.cost = -val_accuracy

            if ant.cost < best_ant.cost:
                best_ant.position = ant.position
                best_ant.cost = ant.cost

        print(f'Generation {generation + 1}, Best Cost: {-best_ant.cost}')

    return best_ant.position
```

```
# Data augmentation and data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
val_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 1379 images belonging to 3 classes.
```

```
validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 344 images belonging to 3 classes.
```

```
# Optimize
best_params = aco_optimize(num_ants=10, num_generations=3)
print('Best Parameters:', best_params)
```

```
Generation 1, Best Cost: 0.9360465407371521
Generation 2, Best Cost: 0.9360465407371521
Generation 3, Best Cost: 0.9360465407371521
Best Parameters: {'filters1': 128, 'kernel_size1': 3, 'filters2': 128, 'kernel_size2': 5, 'dense_units': 512, 'dropout':
```

```
# Evaluate the optimized model
best_model = create_model(best_params)
history = best_model.fit(train_generator, epochs=20, validation_data=validation_generator, verbose=1)
```

```
Epoch 1/20
44/44 [==============================] - 16s 336ms/step - loss: 0.9230 - accuracy: 0.5417 - val_loss: 0.7023 - val_accur
Epoch 2/20
44/44 [==============================] - 16s 367ms/step - loss: 0.6820 - accuracy: 0.7208 - val_loss: 0.5346 - val_accur
Epoch 3/20
44/44 [==============================] - 15s 339ms/step - loss: 0.5345 - accuracy: 0.8071 - val_loss: 0.4236 - val_accur
Epoch 4/20
44/44 [==============================] - 15s 339ms/step - loss: 0.4482 - accuracy: 0.8238 - val_loss: 0.5113 - val_accur
Epoch 5/20
44/44 [==============================] - 15s 343ms/step - loss: 0.3795 - accuracy: 0.8434 - val_loss: 0.2905 - val_accur
Epoch 6/20
44/44 [==============================] - 15s 338ms/step - loss: 0.3195 - accuracy: 0.8753 - val_loss: 0.4642 - val_accur
Epoch 7/20
44/44 [==============================] - 15s 342ms/step - loss: 0.3015 - accuracy: 0.8811 - val_loss: 0.2583 - val_accur
```

```
Epoch 8/20
44/44 [==============================] - 15s 341ms/step - loss: 0.2423 - accuracy: 0.9086 - val_loss: 0.3044 - val_accur
Epoch 9/20
44/44 [==============================] - 15s 341ms/step - loss: 0.2308 - accuracy: 0.9028 - val_loss: 0.2931 - val_accur
Epoch 10/20
44/44 [==============================] - 15s 342ms/step - loss: 0.2022 - accuracy: 0.9268 - val_loss: 0.1957 - val_accur
Epoch 11/20
44/44 [==============================] - 15s 339ms/step - loss: 0.1964 - accuracy: 0.9268 - val_loss: 0.3151 - val_accur
Epoch 12/20
44/44 [==============================] - 15s 343ms/step - loss: 0.1613 - accuracy: 0.9391 - val_loss: 0.1982 - val_accur
Epoch 13/20
44/44 [==============================] - 15s 345ms/step - loss: 0.1856 - accuracy: 0.9239 - val_loss: 0.4261 - val_accur
Epoch 14/20
44/44 [==============================] - 15s 346ms/step - loss: 0.1502 - accuracy: 0.9449 - val_loss: 0.1592 - val_accur
Epoch 15/20
44/44 [==============================] - 15s 343ms/step - loss: 0.1417 - accuracy: 0.9478 - val_loss: 0.2419 - val_accur
Epoch 16/20
44/44 [==============================] - 15s 342ms/step - loss: 0.1706 - accuracy: 0.9398 - val_loss: 0.4235 - val_accur
Epoch 17/20
44/44 [==============================] - 15s 340ms/step - loss: 0.1198 - accuracy: 0.9558 - val_loss: 0.5550 - val_accur
Epoch 18/20
44/44 [==============================] - 15s 342ms/step - loss: 0.1371 - accuracy: 0.9485 - val_loss: 0.2839 - val_accur
Epoch 19/20
44/44 [==============================] - 15s 338ms/step - loss: 0.1270 - accuracy: 0.9529 - val_loss: 0.2764 - val_accur
Epoch 20/20
44/44 [==============================] - 15s 337ms/step - loss: 0.1355 - accuracy: 0.9536 - val_loss: 0.3615 - val_accur
```

```
# Save the model
best_model.save('potato_disease_classifier_aco.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an
  saving_api.save_model(
```

```
# Evaluate the model
val_loss, val_accuracy = best_model.evaluate(validation_generator)
print(f'Validation loss: {val_loss}')
print(f'Validation accuracy: {val_accuracy}')
```

```
11/11 [==============================] - 2s 122ms/step - loss: 0.3615 - accuracy: 0.8750
Validation loss: 0.36153173446655273
Validation accuracy: 0.875
```

```
import os

# Define the path to your directory
directory_path = '/content/drive/MyDrive/potato_dataset/train/Potato___Early_blight'

# List all files in the directory
file_list = os.listdir(directory_path)

# Print the list of files
print(file_list)
```

```
['e00d7014-a909-4d98-892b-da96a2be9a2e___RS_Early.B 7535.JPG', 'c8dedd98-c5d7-4ef4-b4f4-70d0bae0e178___RS_Early.B 7128.J
```

```
# Example: Get the first image file from the list
image_filename = file_list[0]

# Construct the full path
image_path = os.path.join(directory_path, image_filename)

# Print the full path to the image
print(image_path)
```

```
/content/drive/MyDrive/potato_dataset/train/Potato___Early_blight/e00d7014-a909-4d98-892b-da96a2be9a2e___RS_Early.B 7535
```

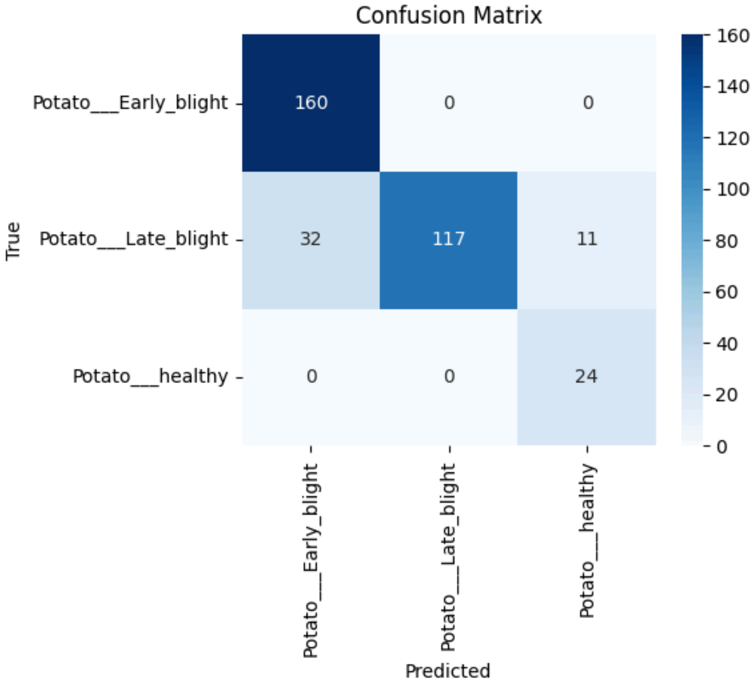
```
import tensorflow as tf
import numpy as np

# Function to preprocess the image
def preprocess_image(image_path, target_size=(150, 150)):
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, axis=0)  # Expand dimensions to create batch of 1
    img_array = img_array / 255.0  # Rescale pixel values
    return img_array

# Load the model on CPU to avoid memory issues
with tf.device('/CPU:0'):
    model = tf.keras.models.load_model('potato_disease_classifier_aco.h5')

# Path to your image
image_path = '/content/drive/MyDrive/potato_dataset/train/Potato___Early_blight/d825093a-2bd7-458d-a9a1-036db6c08dec___RS_Ea

# Preprocess the image
preprocessed_img = preprocess_image(image_path)

# Make prediction
with tf.device('/CPU:0'):
    predictions = model.predict(preprocessed_img)

# Get the class with the highest probability
predicted_class_index = np.argmax(predictions)

# Define the class labels (assuming these are the classes in your dataset)
class_labels = ['Potato___Early_blight', 'Potato___healthy', 'Potato___Late_blight']

# Get the predicted label
predicted_label = class_labels[predicted_class_index]

# Print the predicted label
print(f'Predicted label: {predicted_label}')

# If you want to visualize the image along with the label, you can use matplotlib
import matplotlib.pyplot as plt

# Load the image for visualization
img = tf.keras.preprocessing.image.load_img(image_path)

# Display the image with the predicted label
plt.imshow(img)
plt.title(f'Predicted: {predicted_label}')
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 263ms/step
Predicted label: Potato___Early_blight
```

Predicted: Potato___Early_blight

```
import tensorflow as tf
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the model
with tf.device('/cpu:0'):
    model_path = 'potato_disease_classifier_aco.h5'
    model = tf.keras.models.load_model(model_path)

# Paths to the validation directories
val_dir = '/content/drive/MyDrive/potato_dataset/validation'

# Define the image size and batch size
img_height = 150
img_width = 150
batch_size = 32

# Create a data generator for the validation set
val_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False  # Important to keep the order of images and labels
)

# Ensure that predictions are also done on the CPU
with tf.device('/cpu:0'):
    # Predict on the validation set
    predictions = model.predict(validation_generator)

# Get the predicted class indices
predicted_class_indices = np.argmax(predictions, axis=1)

# Get the true class indices
true_class_indices = validation_generator.classes

# Define the class labels
class_labels = list(validation_generator.class_indices.keys())

# Generate the classification report
report = classification_report(true_class_indices, predicted_class_indices, target_names=class_labels)
print(report)

# Generate the confusion matrix
conf_matrix = confusion_matrix(true_class_indices, predicted_class_indices)

# Plot the confusion matrix
plt.figure(figsize=(5, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
Found 344 images belonging to 3 classes.
11/11 [==============================] - 19s 2s/step
                      precision    recall  f1-score   support

Potato___Early_blight       0.83      1.00      0.91       160
 Potato___Late_blight       1.00      0.73      0.84       160
     Potato___healthy       0.69      1.00      0.81        24

             accuracy                           0.88       344
            macro avg       0.84      0.91      0.86       344
         weighted avg       0.90      0.88      0.87       344
```



ACO+DENSENET

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.optimizers import Adam
import logging
from datetime import datetime
import time
import random


# Define the DenseNet-based model
def create_model(params):
    base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(params['dense_units'], activation='relu')(x)
    x = Dropout(params['dropout'])(x)
    predictions = Dense(3, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    # Freeze the base_model layers during initial training
    for layer in base_model.layers:
        layer.trainable = False

    model.compile(optimizer=Adam(learning_rate=params['learning_rate']),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
class Ant:
    def __init__(self):
        self.position = None
        self.cost = float('inf')

def random_params():
    return {
        'dense_units': np.random.choice([128, 256, 512]),
        'dropout': np.random.uniform(0.2, 0.5),
        'learning_rate': np.random.choice([1e-3, 1e-4, 1e-5])
    }



# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(message)s')

# ACO optimization function with detailed progress tracking
def aco_optimize(num_ants, num_generations):
    best_ant = Ant()
    start_time = time.time()

    for generation in range(num_generations):
        logging.info(f'Starting generation {generation + 1}/{num_generations}')

        ants = [Ant() for _ in range(num_ants)]

        for i, ant in enumerate(ants):
            ant.position = random_params()
            model = create_model(ant.position)

            logging.info(f'Generation {generation + 1}/{num_generations}, Ant {i + 1}/{num_ants}, Starting training')
            start_training_time = time.time()
            history = model.fit(train_generator, epochs=5, validation_data=validation_generator, verbose=1)
            end_training_time = time.time()

            val_accuracy = history.history['val_accuracy'][-1]
            ant.cost = -val_accuracy

            if ant.cost < best_ant.cost:
                best_ant.position = ant.position
                best_ant.cost = ant.cost

            # Print progress for each ant
            logging.info(f'Generation {generation + 1}/{num_generations}, Ant {i + 1}/{num_ants}, Best Cost: {-best_ant.cost
            logging.info(f'Training time for this ant: {(end_training_time - start_training_time)/60:.2f} minutes')

        logging.info(f'Completed generation {generation + 1}/{num_generations}, Best Cost: {-best_ant.cost:.4f}')

        # Estimate remaining time
        elapsed_time = time.time() - start_time
        remaining_generations = num_generations - (generation + 1)
        estimated_time_remaining = (elapsed_time / (generation + 1)) * remaining_generations
        logging.info(f'Elapsed Time: {elapsed_time/60:.2f} minutes, Estimated Time Remaining: {estimated_time_remaining/60:.

    return best_ant.position
```

```
# Image dimensions
img_height = 150
img_width = 150
batch_size = 32

# Data augmentation and data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 1379 images belonging to 3 classes.
Found 344 images belonging to 3 classes.
```

```
# Run the ACO optimization
best_params = aco_optimize(num_ants=10, num_generations=3)
logging.info('Best Parameters: %s', best_params)

# Evaluate the optimized model
best_model = create_model(best_params)
history = best_model.fit(train_generator, epochs=20, validation_data=validation_generator, verbose=1)

# Save the model
best_model.save('potato_disease_classifier_aco_densenet.h5')

# Evaluate the model
val_loss, val_accuracy = best_model.evaluate(validation_generator)
print(f'Validation loss: {val_loss}')
print(f'Validation accuracy: {val_accuracy}')
```

```
Epoch 10/20
44/44 [==============================] - 15s 342ms/step - loss: 0.1388 - accuracy: 0.9478 - val_loss: 0.0758 - val_accur
Epoch 11/20
44/44 [==============================] - 15s 344ms/step - loss: 0.1365 - accuracy: 0.9478 - val_loss: 0.0835 - val_accur
Epoch 12/20
44/44 [==============================] - 15s 345ms/step - loss: 0.1484 - accuracy: 0.9507 - val_loss: 0.0651 - val_accur
Epoch 13/20
44/44 [==============================] - 15s 343ms/step - loss: 0.1267 - accuracy: 0.9507 - val_loss: 0.0873 - val_accur
Epoch 14/20
44/44 [==============================] - 15s 343ms/step - loss: 0.1742 - accuracy: 0.9376 - val_loss: 0.0721 - val_accur
Epoch 15/20
44/44 [==============================] - 15s 345ms/step - loss: 0.1345 - accuracy: 0.9500 - val_loss: 0.0721 - val_accur
Epoch 16/20
44/44 [==============================] - 15s 344ms/step - loss: 0.1259 - accuracy: 0.9521 - val_loss: 0.1102 - val_accur
Epoch 17/20
44/44 [==============================] - 16s 356ms/step - loss: 0.1352 - accuracy: 0.9427 - val_loss: 0.0646 - val_accur
Epoch 18/20
44/44 [==============================] - 15s 347ms/step - loss: 0.1499 - accuracy: 0.9427 - val_loss: 0.0977 - val_accur
Epoch 19/20
44/44 [==============================] - 15s 347ms/step - loss: 0.1175 - accuracy: 0.9565 - val_loss: 0.1011 - val_accur
Epoch 20/20
44/44 [==============================] - 15s 344ms/step - loss: 0.1079 - accuracy: 0.9666 - val_loss: 0.0661 - val_accur
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an
  saving_api.save_model(
11/11 [==============================] - 2s 142ms/step - loss: 0.0661 - accuracy: 0.9767
Validation loss: 0.06613235920667648
Validation accuracy: 0.9767441749572754
```

```python
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os

# Load the model
model_path = 'potato_disease_classifier_aco_densenet.h5'
model = tf.keras.models.load_model(model_path)

# Define the image size
img_height = 150
img_width = 150

# Define the validation data generator
val_dir = '/content/drive/MyDrive/potato_dataset/validation'

val_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=1,  # Process one image at a time
    class_mode='categorical',
    shuffle=False  # Important to keep the order of images and labels
)

# Get the class indices and corresponding labels
class_indices = validation_generator.class_indices
class_labels = list(class_indices.keys())

# Get a single image and its label
img_path = validation_generator.filepaths[0]
true_label = img_path.split('/')[-2]  # Extract the true label from the directory name

# Preprocess the image
img = tf.keras.preprocessing.image.load_img(img_path, target_size=(img_height, img_width))
img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Create batch dimension
img_array /= 255.0  # Rescale the image

# Predict the label
predictions = model.predict(img_array)
predicted_class_index = np.argmax(predictions, axis=1)[0]
predicted_label = class_labels[predicted_class_index]

# Plot the image with its true and predicted labels
plt.imshow(img)
plt.title(f'True: {true_label}, Predicted: {predicted_label}')
plt.axis('off')
plt.show()
```

```
Found 344 images belonging to 3 classes.
1/1 [==============================] - 7s 7s/step
```

True: Potato___Early_blight, Predicted: Potato___Early_blight

```
import tensorflow as tf
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the model
with tf.device('/cpu:0'):
    model_path = 'potato_disease_classifier_aco_densenet.h5'
    model = tf.keras.models.load_model(model_path)

# Paths to the validation directories
val_dir = '/content/drive/MyDrive/potato_dataset/validation'
```