In [1]:
```
# Importing python modules
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import tensorflow as tf
```

In [2]:
```

data_dir = "data"
batch_size = 32
img_height = 128
img_width = 128

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)
```

```
Found 2152 files belonging to 3 classes.
Using 1722 files for training.
Found 2152 files belonging to 3 classes.
Using 430 files for validation.
```

In [3]:
```
# Configure the dataset for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

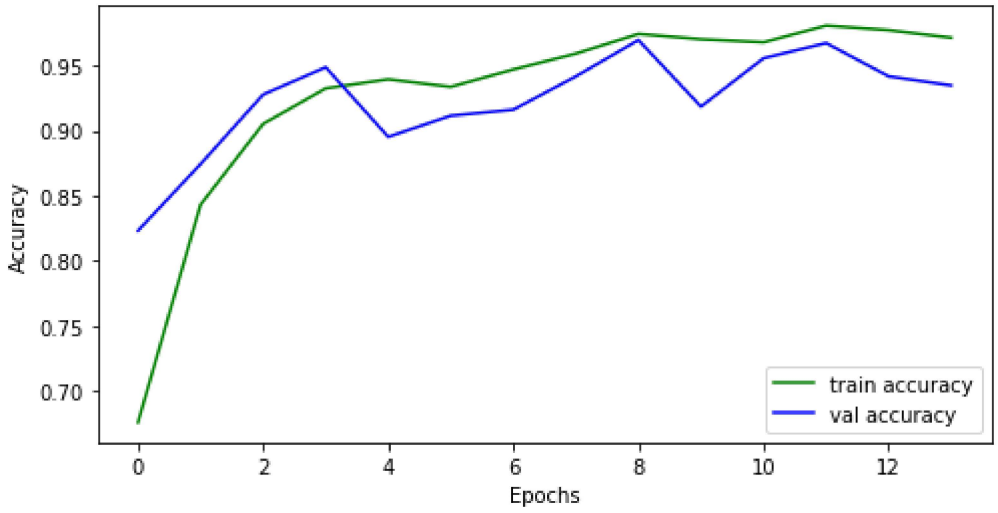
In [4]:

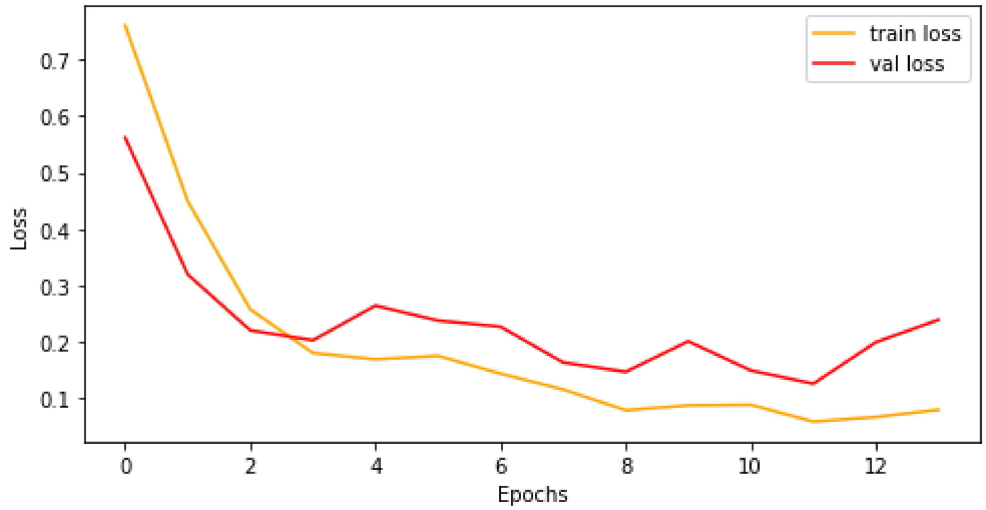
```
# CNN Model architecture

num_classes = 3

model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(img_height, img_width, 3)),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255),

    tf.keras.layers.Conv2D(16, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(32, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Conv2D(64, 3, activation='relu'),
    tf.keras.layers.MaxPooling2D(),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

In [5]:
```
# Model compilation
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Early stopping to avoid overfitting
earlystop_callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',
                                                      min_delta=0.0001,
                                                      patience=5)
# training the model
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=20,
                    callbacks=[earlystop_callback])

# Save the model
model.save("model_v1.h5")
```

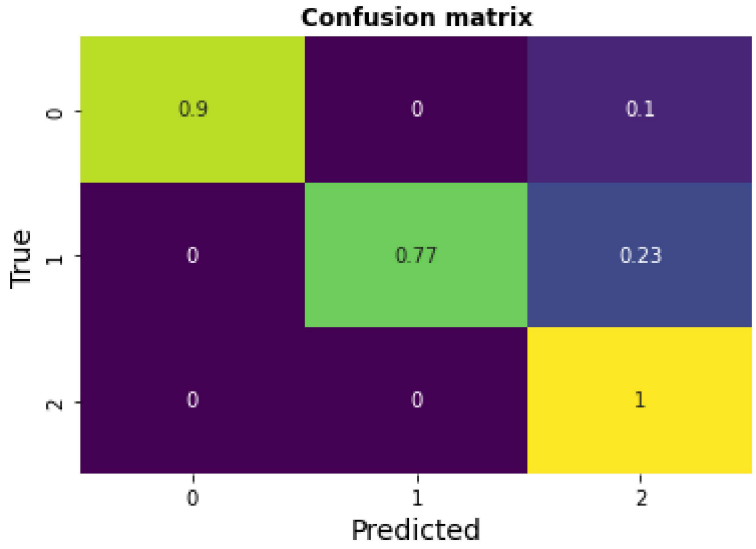
```
Epoch 1/20
54/54 [==============================] - 23s 388ms/step - loss: 0.7607 - accu
racy: 0.6760 - val_loss: 0.5616 - val_accuracy: 0.8233
Epoch 2/20
54/54 [==============================] - 16s 296ms/step - loss: 0.4487 - accu
racy: 0.8432 - val_loss: 0.3189 - val_accuracy: 0.8744
Epoch 3/20
54/54 [==============================] - 15s 272ms/step - loss: 0.2569 - accu
racy: 0.9053 - val_loss: 0.2197 - val_accuracy: 0.9279
Epoch 4/20
54/54 [==============================] - 15s 284ms/step - loss: 0.1795 - accu
racy: 0.9326 - val_loss: 0.2021 - val_accuracy: 0.9488
Epoch 5/20
54/54 [==============================] - 16s 295ms/step - loss: 0.1682 - accu
racy: 0.9396 - val_loss: 0.2636 - val_accuracy: 0.8953
Epoch 6/20
54/54 [==============================] - 16s 301ms/step - loss: 0.1743 - accu
racy: 0.9338 - val_loss: 0.2370 - val_accuracy: 0.9116
Epoch 7/20
54/54 [==============================] - 17s 307ms/step - loss: 0.1429 - accu
racy: 0.9472 - val_loss: 0.2262 - val_accuracy: 0.9163
Epoch 8/20
54/54 [==============================] - 16s 293ms/step - loss: 0.1147 - accu
racy: 0.9593 - val_loss: 0.1627 - val_accuracy: 0.9419
Epoch 9/20
54/54 [==============================] - 16s 296ms/step - loss: 0.0779 - accu
racy: 0.9744 - val_loss: 0.1461 - val_accuracy: 0.9698
Epoch 10/20
54/54 [==============================] - 15s 280ms/step - loss: 0.0864 - accu
racy: 0.9704 - val_loss: 0.2003 - val_accuracy: 0.9186
Epoch 11/20
54/54 [==============================] - 16s 302ms/step - loss: 0.0876 - accu
racy: 0.9681 - val_loss: 0.1485 - val_accuracy: 0.9558
Epoch 12/20
54/54 [==============================] - 16s 303ms/step - loss: 0.0577 - accu
racy: 0.9808 - val_loss: 0.1249 - val_accuracy: 0.9674
Epoch 13/20
54/54 [==============================] - 17s 317ms/step - loss: 0.0658 - accu
racy: 0.9774 - val_loss: 0.1988 - val_accuracy: 0.9419
Epoch 14/20
54/54 [==============================] - 17s 310ms/step - loss: 0.0787 - accu
racy: 0.9715 - val_loss: 0.2384 - val_accuracy: 0.9349
```

In [6]:
```
train_loss = history.history['loss']
train_acc = history.history['accuracy']
valid_loss = history.history['val_loss']
valid_acc = history.history['val_accuracy']

# Accuracy plots
plt.figure(figsize=(8, 4))
plt.plot(train_acc, color='green', linestyle='-', label='train accuracy')
plt.plot(valid_acc, color='blue', linestyle='-', label='val accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# loss plots
plt.figure(figsize=(8, 4))
plt.plot(train_loss, color='orange', linestyle='-', label='train loss')
plt.plot(valid_loss, color='red', linestyle='-', label='val loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





In [7]:
```
y_pred = []  # store predicted labels
y_true = []  # store true labels

# iterate over the dataset
for image_batch, label_batch in val_ds:
   # append true labels
   y_true.append(label_batch)
   # compute predictions
   preds = model.predict(image_batch)
   # append predicted labels
   y_pred.append(np.argmax(preds, axis = - 1))

# convert the true and predicted labels into tensors
correct_labels = tf.concat([item for item in y_true], axis = 0)
predicted_labels = tf.concat([item for item in y_pred], axis = 0)

# Confusion matrix
cm = confusion_matrix(correct_labels, predicted_labels, normalize='true')

sns.heatmap(cm, annot=True, cmap='viridis', cbar=None)

plt.title("Confusion matrix", fontweight='bold')
plt.ylabel("True", fontsize=14)
plt.xlabel("Predicted", fontsize=14)

plt.show()
```



In [8]:
```
print(classification_report(correct_labels, predicted_labels))
```

```
              precision    recall  f1-score   support

           0       1.00      0.90      0.95       204
           1       1.00      0.77      0.87        31
           2       0.87      1.00      0.93       195

    accuracy                           0.93       430
   macro avg       0.96      0.89      0.92       430
weighted avg       0.94      0.93      0.93       430
```

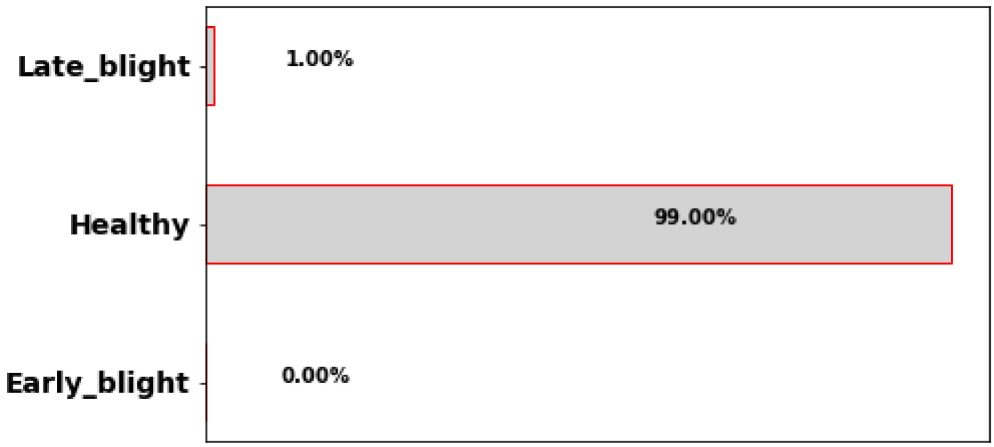
In [9]:
```
model.save(r"C:\Users\gadda\plant-disease-classifier\application\static\models
```

In [10]:
```
def prediction(img):
    class_names = ['Early_blight', 'Healthy', 'Late_blight']

    my_image = load_img(img, target_size=(img_height, img_width))
    my_image = img_to_array(my_image)
    my_image = np.expand_dims(my_image, 0)

    out = np.round(model.predict(my_image)[0], 2)
    fig = plt.figure(figsize=(7, 4))
    plt.barh(class_names, out, color='lightgray',edgecolor='red', linewidth=1,

    for index, value in enumerate(out):
        plt.text(value/2 + 0.1, index, f"{100*value:.2f}%", fontweight='bold')

    plt.xticks([])
    plt.yticks([0, 1, 2], labels=class_names, fontweight='bold', fontsize=14)
    fig.savefig('pred_img.png', bbox_inches='tight')
    return plt.show()
```

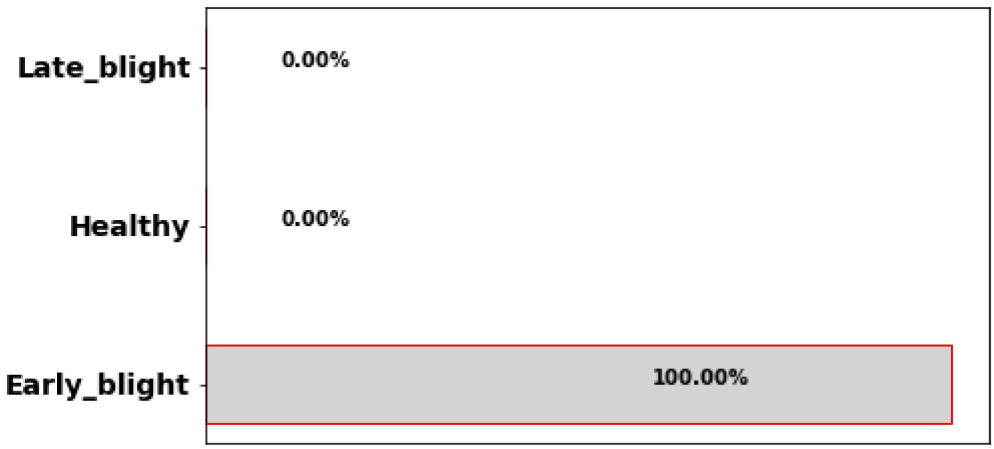
In [ ]:
```
```

In [11]:
```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Rest of your code
```

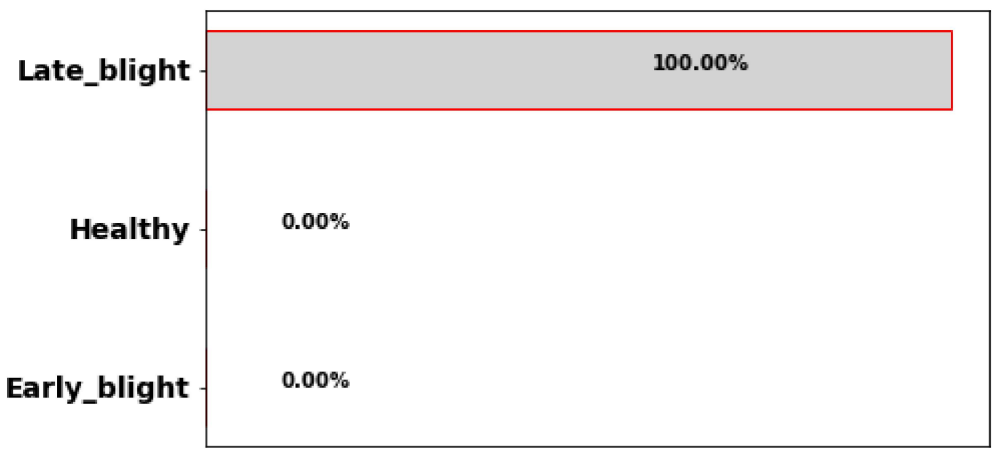
In [12]:
```
####### Prediction on single Image
img = r"C:\Users\gadda\data\Healthy\04481ca2-f94c-457e-b785-1ac05800b7ec___RS_
prediction(img)
```



In [13]:
```
####### Prediction on single Image
img = r"C:\Users\gadda\data\Early_blight\0182e991-97f0-4805-a1f7-6e1b4306d518_
prediction(img)
```



In [14]:
```
####### Prediction on single Image
img = r"C:\Users\gadda\data\Late_blight\01270f5c-a44b-4da7-9398-289088c197ab__
prediction(img)
```

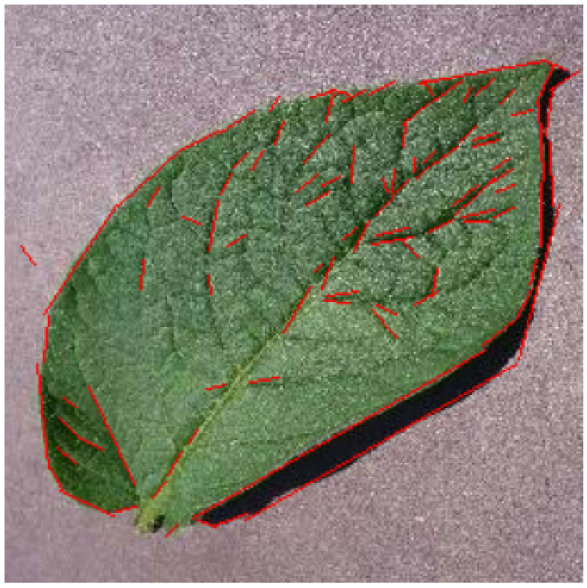


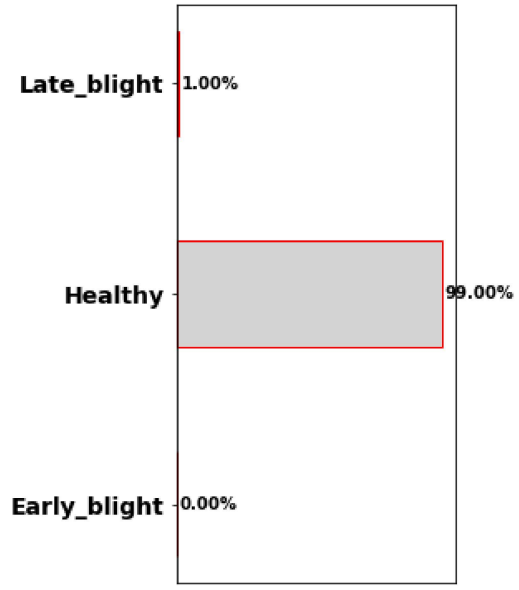
In [15]:

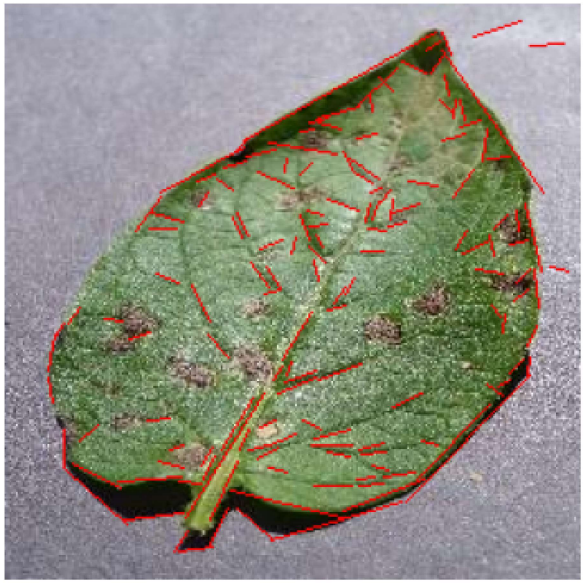
```
# Evaluate model on test dataset
test_loss, test_accuracy = model.evaluate(val_ds)

print(f"Test accuracy: {test_accuracy}")
```

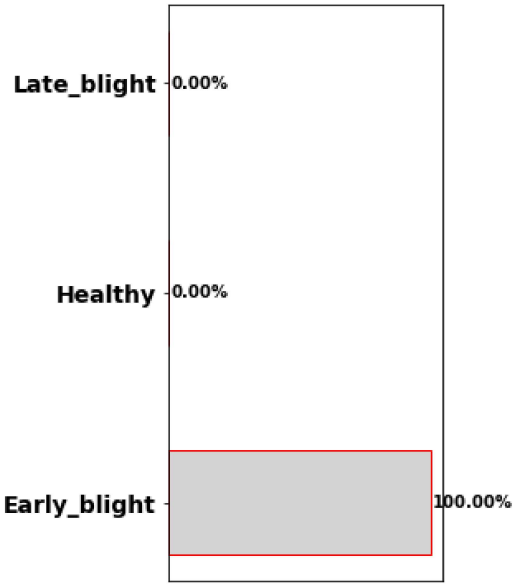
```
14/14 [==============================] - 1s 77ms/step - loss: 0.2384 - accura
cy: 0.9349
Test accuracy: 0.934883713722229
```

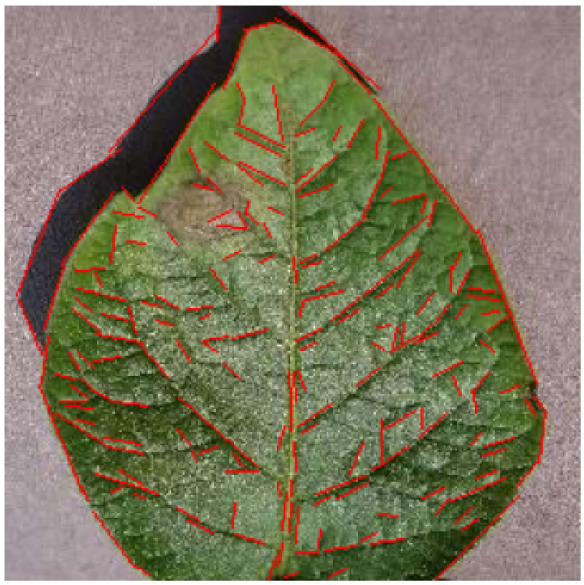
In [16]:
```
import cv2
def prediction(img):
    class_names = ['Early_blight', 'Healthy', 'Late_blight']

    my_image = load_img(img, target_size=(img_height, img_width))
    my_image_array = img_to_array(my_image)
    my_image_array = np.expand_dims(my_image_array, 0)

    out = np.round(model.predict(my_image_array)[0], 2)

    # Create subplots with adjusted spacing
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), gridspec_kw={'width_

    # Plot image with LSD visualization
    img_cv2 = cv2.imread(img)
    gray = cv2.cvtColor(img_cv2, cv2.COLOR_BGR2GRAY)
    lsd = cv2.createLineSegmentDetector(0)
    lines = lsd.detect(gray)[0]
    drawn_img = lsd.drawSegments(img_cv2, lines)
    ax1.imshow(cv2.cvtColor(drawn_img, cv2.COLOR_BGR2RGB))
    ax1.axis('off')  # Hide axis for the image

    # Plot bar chart
    bar_positions = np.arange(len(class_names))
    ax2.barh(bar_positions, out, color='lightgray', edgecolor='red', linewidth

    for i, v in enumerate(out):
        ax2.text(v + 0.01, i, f"{100 * v:.2f}%", color='black', va='center', f

    ax2.set_xticks([])
    ax2.set_yticks(bar_positions)
    ax2.set_yticklabels(class_names, fontweight='bold', fontsize=14)

    # Adjust spacing between subplots
    fig.tight_layout(pad=4.0)

    fig.savefig('pred_img.png', bbox_inches='tight')
    return plt.show()

# Rest of your code

# Call the prediction function with the appropriate image paths
img1 = r"C:\Users\gadda\data\Healthy\04481ca2-f94c-457e-b785-1ac05800b7ec___RS_
prediction(img1)

img2 = r"C:\Users\gadda\data\Early_blight\0182e991-97f0-4805-a1f7-6e1b4306d518_
prediction(img2)

img3 = r"C:\Users\gadda\data\Late_blight\01270f5c-a44b-4da7-9398-289088c197ab_
prediction(img3)
```

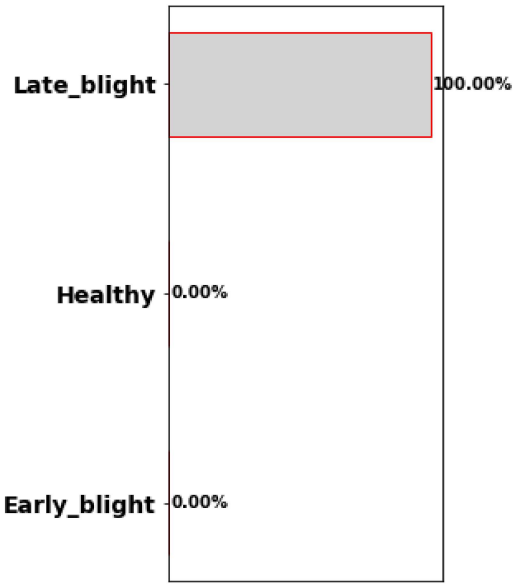



Late_blight
1.00%
Healthy
99.00%
Early_blight
0.00%




Late_blight
0.00%
Healthy
0.00%
Early_blight
100.00%




Late_blight
100.00%
Healthy
0.00%
Early_blight
0.00%

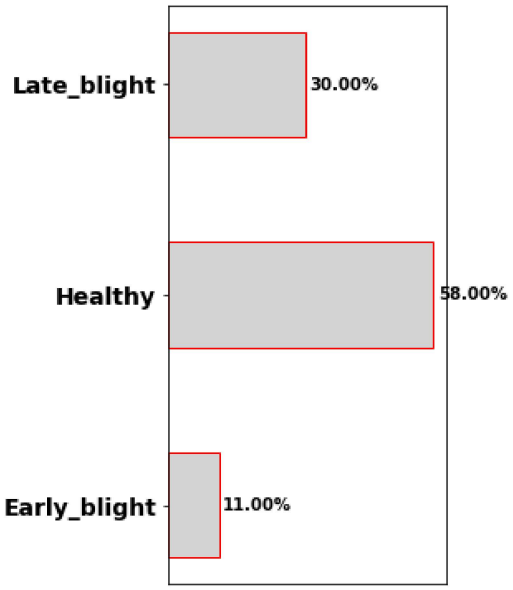
In [17]:
```
from sklearn.metrics import accuracy_score

def evaluate_model_accuracy(model, test_dataset):
    y_true = []
    y_pred = []

    for image_batch, label_batch in test_dataset:
        y_true.extend(label_batch.numpy())
        preds = model.predict(image_batch)
        y_pred.extend(np.argmax(preds, axis=-1))

    acc = accuracy_score(y_true, y_pred)
    return acc

# Assuming you have a test dataset
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

# Evaluate the model's accuracy
model_accuracy = evaluate_model_accuracy(model, test_ds)
print(f"Model Accuracy: {model_accuracy}")
```

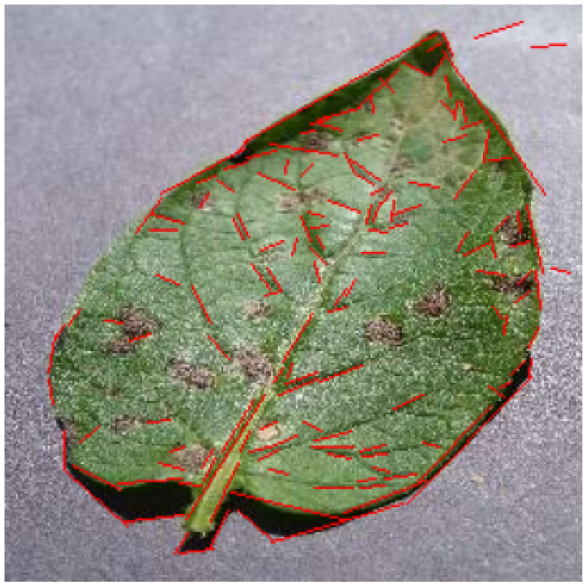
```
Found 2152 files belonging to 3 classes.
Using 430 files for validation.
Model Accuracy: 0.9348837209302325
```

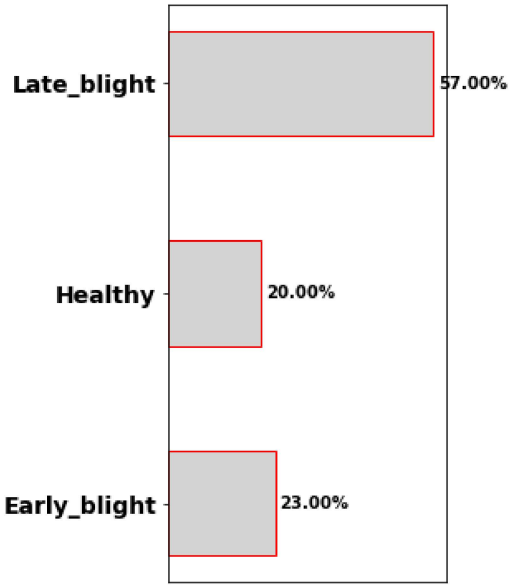
In [ ]:

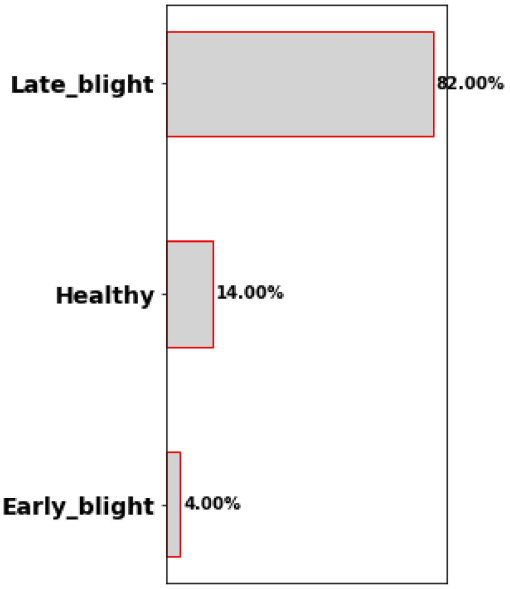
In [18]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Flatten, Reshape
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# Define the image dimensions
img_height, img_width = 128, 128

# Define the class names
class_names = ['Early_blight', 'Healthy', 'Late_blight']

# Define LSTM model
model = Sequential()
model.add(Reshape((img_height, img_width * 3), input_shape=(img_height, img_wi
model.add(LSTM(100))
model.add(Dense(len(class_names), activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc

# Function to preprocess image
def preprocess_img(img_path):
    my_image = load_img(img_path, target_size=(img_height, img_width))
    my_image_array = img_to_array(my_image)
    return my_image_array

# Function for prediction and visualization
def prediction_lstm(img_path):
    my_image_array = preprocess_img(img_path)
    my_image_array = np.expand_dims(my_image_array, axis=0)

    # Make predictions
    out = np.round(model.predict(my_image_array)[0], 2)

    # Create subplots with adjusted spacing
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), gridspec_kw={'width_

    # Plot image with LSD visualization
    img_cv2 = cv2.imread(img_path)
    gray = cv2.cvtColor(img_cv2, cv2.COLOR_BGR2GRAY)
    lsd = cv2.createLineSegmentDetector(0)
    lines = lsd.detect(gray)[0]
    drawn_img = lsd.drawSegments(img_cv2, lines)
    ax1.imshow(cv2.cvtColor(drawn_img, cv2.COLOR_BGR2RGB))
    ax1.axis('off')  # Hide axis for the image

    # Plot bar chart
    bar_positions = np.arange(len(class_names))
    ax2.barh(bar_positions, out, color='lightgray', edgecolor='red', linewidth

    for i, v in enumerate(out):
        ax2.text(v + 0.01, i, f"{100 * v:.2f}%", color='black', va='center', f

    ax2.set_xticks([])
    ax2.set_yticks(bar_positions)
    ax2.set_yticklabels(class_names, fontweight='bold', fontsize=14)
```

```
    # Adjust spacing between subplots
    fig.tight_layout(pad=4.0)

    fig.savefig('pred_img.png', bbox_inches='tight')
    return plt.show()

# Example usage with appropriate image paths
img1 = r"C:\Users\gadda\data\Healthy\04481ca2-f94c-457e-b785-1ac05800b7ec___RS_
img2 = r"C:\Users\gadda\data\Early_blight\0182e991-97f0-4805-a1f7-6e1b4306d518_
img3 = r"C:\Users\gadda\data\Late_blight\01270f5c-a44b-4da7-9398-289088c197ab_

prediction_lstm(img1)
prediction_lstm(img2)
prediction_lstm(img3)
```









In [19]:
```
# Assuming you have a test dataset with labeled images
test_img_paths = [img1, img2, img3]
test_labels = [1, 0, 2]
# Assuming the labels are 0, 1, 2 for ['Early_blight', 'Healthy', 'Late_blight

# Initialize an empty list to store the predictions
predictions = []

# Iterate through the test images and make predictions
for img_path in test_img_paths:
    my_image_array = preprocess_img(img_path)
    my_image_array = np.expand_dims(my_image_array, axis=0)
    out = np.round(model.predict(my_image_array)[0], 2)
    predicted_label = np.argmax(out)
    predictions.append(predicted_label)

# Compare the predicted labels with the true labels and compute the accuracy
num_correct = sum([1 for i in range(len(test_labels)) if test_labels[i] == pre
accuracy = num_correct / len(test_labels)
print(f"Accuracy of the LSTM model: {accuracy:.2f}")
```

```
Accuracy of the LSTM model: 0.67
```

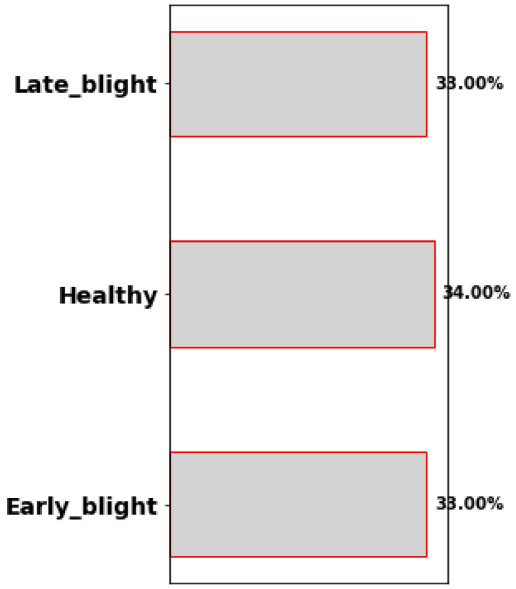
In [20]:
```
pip install https://github.com/matterport/Mask_RCNN/archive/master.zip
```

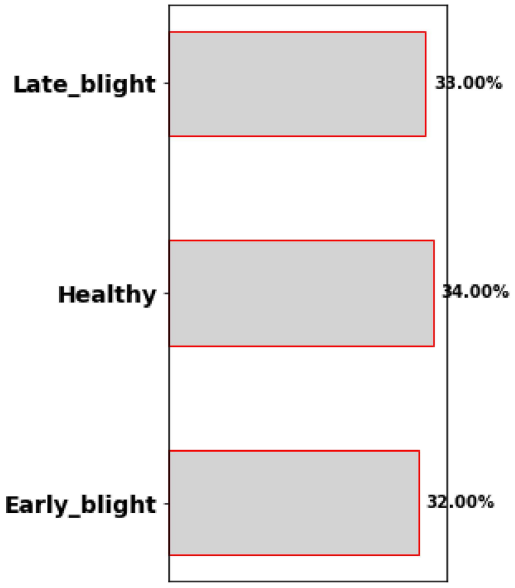
```
Collecting https://github.com/matterport/Mask_RCNN/archive/master.zip (http
s://github.com/matterport/Mask_RCNN/archive/master.zip)
  Using cached https://github.com/matterport/Mask_RCNN/archive/master.zip (ht
tps://github.com/matterport/Mask_RCNN/archive/master.zip)
Note: you may need to restart the kernel to use updated packages.
```

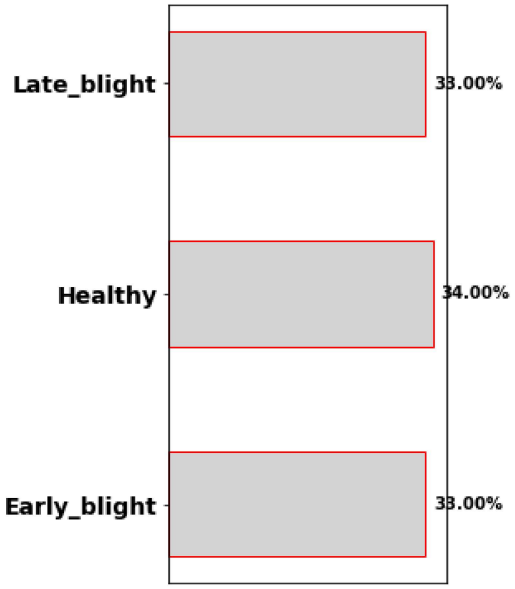
In [21]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Batc
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.optimizers import Adam

# Define the image dimensions
img_height, img_width = 128, 128

# Define the class names
class_names = ['Early_blight', 'Healthy', 'Late_blight']

# Define SCNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_w
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(len(class_names), activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.0001), metr

# Function to preprocess image
def preprocess_img(img_path):
    my_image = load_img(img_path, target_size=(img_height, img_width))
    my_image_array = img_to_array(my_image) / 255.0  # Normalize pixel values
    return my_image_array

# Function for prediction and visualization
def prediction_scnn(img_path):
    my_image_array = preprocess_img(img_path)
    my_image_array = np.expand_dims(my_image_array, axis=0)

    # Make predictions
    out = np.round(model.predict(my_image_array)[0], 2)

    # Create subplots with adjusted spacing
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), gridspec_kw={'width_

    # Plot image
    img_cv2 = cv2.imread(img_path)
    img_rgb = cv2.cvtColor(img_cv2, cv2.COLOR_BGR2RGB)
    ax1.imshow(img_rgb)
    ax1.axis('off')  # Hide axis for the image

    # Plot bar chart
```

```
    bar_positions = np.arange(len(class_names))
    ax2.barh(bar_positions, out, color='lightgray', edgecolor='red', linewidth

    for i, v in enumerate(out):
        ax2.text(v + 0.01, i, f"{100 * v:.2f}%", color='black', va='center', f

    ax2.set_xticks([])
    ax2.set_yticks(bar_positions)
    ax2.set_yticklabels(class_names, fontweight='bold', fontsize=14)

    # Adjust spacing between subplots
    fig.tight_layout(pad=4.0)

    plt.show()

# Example usage with appropriate image paths
img1 = r"C:\Users\gadda\data\Healthy\04481ca2-f94c-457e-b785-1ac05800b7ec___RS
img2 = r"C:\Users\gadda\data\Early_blight\0182e991-97f0-4805-a1f7-6e1b4306d518
img3 = r"C:\Users\gadda\data\Late_blight\01270f5c-a44b-4da7-9398-289088c197ab_

# Make predictions for the example images
prediction_scnn(img1)
prediction_scnn(img2)
prediction_scnn(img3)
```

```
C:\Users\gadda\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:105: Us
erWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```




Late_blight
33.00%
Healthy
34.00%
Early_blight
32.00%


Late_blight
33.00%
Healthy
34.00%
Early_blight
33.00%

In [ ]:

In [ ]: