```
# from __future__ import print_function, division
import os,random

import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.axes_grid1 import AxesGrid
from tqdm.notebook import tqdm
import cv2

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms

from sklearn.metrics import (
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix

)

import time
import copy
```

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
%cd drive/My\ Drive
!cp data.zip /content/
```

```
/content/drive/My Drive
```

```
%cd /content/
!unzip data.zip
```

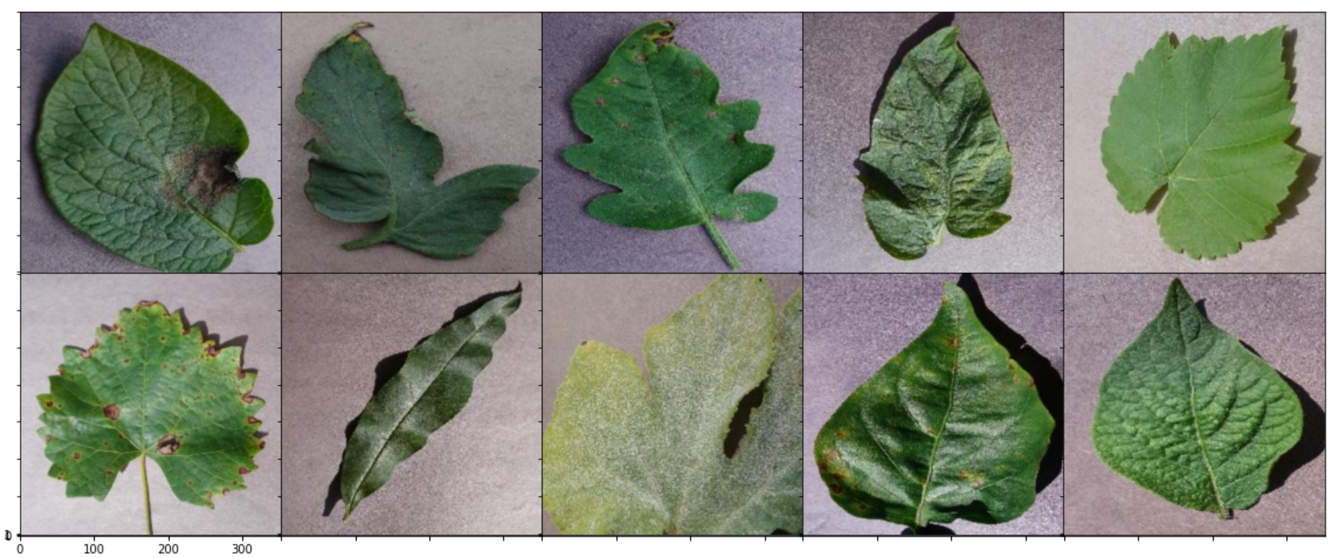
```
  inflating: content/data/val/Potato___Early_blight/b301c9cd-7624-4261-ab4f-962936eb
  inflating: content/data/val/Potato___Early_blight/68662e76-300e-49f6-8467-27dd63aa
  inflating: content/data/val/Potato___Early_blight/b512ab1a-0ea5-47d6-b998-7c9f32ed
  inflating: content/data/val/Potato___Early_blight/2c48971d-9f73-401d-a62a-f3729d08
  inflating: content/data/val/Potato___Early_blight/7fb3dc5c-f90b-46fe-8eb6-2d210389
  inflating: content/data/val/Potato___Early_blight/4f71ba26-6612-427b-8dc0-4d8cc909
  inflating: content/data/val/Potato___Early_blight/b6220993-c51f-48fa-bee9-fb5cb89c
```

```
inflating: content/data/val/Potato___Early_blight/0604174e-3018-4faa-9975-0be32d2c
inflating: content/data/val/Potato___Early_blight/6319585f-ed2f-4657-ae79-a23db65d
inflating: content/data/val/Potato___Early_blight/3aea17a1-9413-4312-bcf2-e9aadb73
inflating: content/data/val/Potato___Early_blight/848336c4-eaad-47e7-8458-5e80d552
inflating: content/data/val/Potato___Early_blight/5c3363d5-5873-40f5-b179-37ab6df0
inflating: content/data/val/Potato___Early_blight/8df7a062-73d6-468e-8354-baa641dd
inflating: content/data/val/Potato___Early_blight/9e092cd4-dcd4-4f9b-a901-48ac13a6
inflating: content/data/val/Potato___Early_blight/0faca7fe-7254-4dfa-8388-bbc77633
inflating: content/data/val/Potato___Early_blight/f686133a-e89a-4242-a52d-02f32ffd
inflating: content/data/val/Potato___Early_blight/b1b3c9c7-a4b9-4048-be45-fa9c5e50
inflating: content/data/val/Potato___Early_blight/0267d4ca-522e-4ca0-b1a2-ce925e5b
inflating: content/data/val/Potato___Early_blight/c3d68a2f-0c7a-4060-bfc4-8baf01d3
inflating: content/data/val/Potato___Early_blight/334fd34b-f4aa-4cc2-9ac9-8b85df65
inflating: content/data/val/Potato___Early_blight/67468907-b28a-4740-a9f4-ce69b938
inflating: content/data/val/Potato___Early_blight/0d2e2971-f1c9-4278-b35c-91dd8a22

inflating: content/data/val/Potato___Early_blight/99821810-926a-4ac4-9bc7-cbae991f
inflating: content/data/val/Potato___Early_blight/d1b4cb77-db0e-42db-b1c0-25d22284
inflating: content/data/val/Potato___Early_blight/a0d8a499-e9e4-4c88-829c-7c227007
inflating: content/data/val/Potato___Early_blight/0a0744dc-8486-4fbb-a44b-4d63e6db
inflating: content/data/val/Potato___Early_blight/bc378ba0-533d-42db-a8b2-82decc73
inflating: content/data/val/Potato___Early_blight/907f26c5-5996-41b3-877b-7de61226
inflating: content/data/val/Potato___Early_blight/1767fee7-18fd-4597-8c77-d41ec2d6
inflating: content/data/val/Potato___Early_blight/82611670-959a-401b-8ed0-4b4fdad5
inflating: content/data/val/Potato___Early_blight/ed270d5d-3523-4bc2-b208-4f5304bb
inflating: content/data/val/Potato___Early_blight/7d90df0a-da17-4641-9006-75aa7de6
inflating: content/data/val/Potato___Early_blight/5cb0b99b-2e14-43b3-aa51-51a7ce66
inflating: content/data/val/Potato___Early_blight/e0326fde-a613-4ec2-bc3b-e1d2154b
inflating: content/data/val/Potato___Early_blight/a2fe15c2-1900-4fbd-9e69-771b1693
inflating: content/data/val/Potato___Early_blight/bb4e9c01-6166-424d-a379-9ef405fd
inflating: content/data/val/Potato___Early_blight/232c8d25-3bfe-41ea-b24f-2b3629a0
inflating: content/data/val/Potato___Early_blight/f3b29f09-c337-4233-8968-0fcc66c9
inflating: content/data/val/Potato___Early_blight/ab0878eb-0310-4788-b2d7-14102cdd
inflating: content/data/val/Potato___Early_blight/f5fca019-b28f-4000-9d5e-b0b5fd06
inflating: content/data/val/Potato___Early_blight/8784362f-5fc6-4ad9-b11d-d06b272c
inflating: content/data/val/Potato___Early_blight/db60c4cf-30f1-460c-a345-3045f192
inflating: content/data/val/Potato___Early_blight/53de9716-fcc5-4241-9e8d-21792a5c
inflating: content/data/val/Potato___Early_blight/5fd6c76a-1e89-4991-9991-2b61670d
inflating: content/data/val/Potato___Early_blight/4ab9d59b-57ee-4e22-a6bb-d8152710
inflating: content/data/val/Potato___Early_blight/96ab8c2b-9067-4af7-821c-b14a3c7e
inflating: content/data/val/Potato___Early_blight/094fbf4c-da00-4037-82af-03e712d8
inflating: content/data/val/Potato___Early_blight/719dcce1-73f0-42c8-9941-44eb2b23
inflating: content/data/val/Potato___Early_blight/57c5fe0f-ce93-412a-a29e-a70a4aa2
inflating: content/data/val/Potato___Early_blight/b09e0cba-0564-4dde-a90d-3c0d90d6
inflating: content/data/val/Potato___Early_blight/fbbe7b93-e6da-4dc8-b1ee-ba5646a5
inflating: content/data/val/Potato___Early_blight/f6d8c094-970c-41c5-9f5d-3ded7c8f
inflating: content/data/val/Potato___Early_blight/b2b7758e-f201-4d9c-bc27-d53bb09b
inflating: content/data/val/Potato___Early_blight/38a72c90-ed53-4432-b625-01c1f962
inflating: content/data/val/Potato___Early_blight/91a21acf-c95a-4ffc-90f2-8c64456f
inflating: content/data/val/Potato___Early_blight/4be99b40-e269-4c69-8c90-ac755146
 creating: content/data/.ipynb_checkpoints/
```

```
!rm -rf /content/data/.ipynb_checkpoints
```

```
path = r"data/train"
```

```
random_filenames = []
for tag in os.listdir(path):
    random_filenames.append(path+"/"+tag+"/"+random.choice([
        x for x in os.listdir(os.path.join(path,tag))
        if os.path.isfile(os.path.join(path,tag, x))
    ]))

grid = AxesGrid(plt.figure(1, (20,20)), 111, nrows_ncols=(4, 5), axes_pad=0, label_mode="1")

i = 0
for img_name in random_filenames[0:10]:

    # Download image
    image = cv2.imread(img_name)
    image = cv2.resize(image, (352, 352))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Show image in grid
    grid[i].imshow(image)
    i = i+1
```




```
def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0
```

```
    best_acc = 0.0
    accuracy_stats = {'train': [], "val": []}
    loss_stats = {'train': [], "val": []}

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            print("Reached first for loop:",phase)
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            print("phase:", phase, "runningLoss:", running_loss, "datasetSize", dataset_sizes
            print("preds:", preds, "\n")
            print("labels:", labels, "\n")
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            loss_stats[phase].append(epoch_loss)
```

```
            accuracy_stats[phase].append(np.float(epoch_acc))


            print("Loss = {}\n".format(np.float(epoch_loss)))
            print("Acc = {}\n".format(np.float(epoch_acc)))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

#         print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model, accuracy_stats, loss_stats
```

```


data_path = r"data"

### Prepare the dataset
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

image_datasets = {x: datasets.ImageFolder(os.path.join(data_path, x), data_transforms[x]) for
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4, shuffle=True,
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")



model_ft = models.mobilenet_v2(pretrained=True)
```

```
num_ftrs = model_ft.classifier[1].in_features
model_ft.classifier[1] = nn.Linear(num_ftrs, len(class_names))

model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
model_ft, accuracy_stats, loss_stats = train_model(model_ft, criterion, optimizer_ft, exp_lr_

# Save the model
torch.save(model_ft, '/content/model.pth')

# Save the labels
with open('/content/labels.txt', 'w') as f:
    f.writelines(["%s\n" % item  for item in class_names])
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" to /root/.
100%                                        13.6M/13.6M [00:10<00:00, 1.36MB/s]

Epoch 0/9
----------
Reached first for loop: train
phase: train runningLoss: 5088.492938789481 datasetSize 13851
preds: tensor([ 5,  6, 12], device='cuda:0')

labels: tensor([ 5,  6, 12], device='cuda:0')

Loss = 0.36737368701100864

Acc = 0.884990253411306

Reached first for loop: val
phase: val runningLoss: 239.297467057404 datasetSize 3534
preds: tensor([1, 9], device='cuda:0')

labels: tensor([1, 9], device='cuda:0')

Loss = 0.06771292220073684

Acc = 0.9779286926994907

Epoch 1/9
----------
Reached first for loop: train
phase: train runningLoss: 2194.630438374181 datasetSize 13851
preds: tensor([ 3,  4, 10], device='cuda:0')

labels: tensor([ 3,  4, 10], device='cuda:0')

Loss = 0.15844563124497732

Acc = 0.954371525521623

Reached first for loop: val
phase: val runningLoss: 521.5361849306519 datasetSize 3534
preds: tensor([10, 12], device='cuda:0')

labels: tensor([10, 12], device='cuda:0')

Loss = 0.1475767359735857

Acc = 0.9468024900962083

Epoch 2/9
----------
Reached first for loop: train
phase: train runningLoss: 1471.1728766376837 datasetSize 13851
preds: tensor([ 3,  6, 12], device='cuda:0')

labels: tensor([ 3,  6, 12], device='cuda:0')

Loss = 0.1062141994540238
```

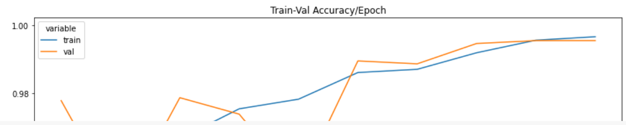
```
Acc = 0.9667894014872572

Reached first for loop: val
phase: val runningLoss: 264.4859655717703 datasetSize 3534
preds: tensor([7, 3], device='cuda:0')

labels: tensor([7, 3], device='cuda:0')

Loss = 0.07484039772828814

Acc = 0.9787775891341257

Epoch 3/9
----------
Reached first for loop: train
phase: train runningLoss: 1171.4279195938943 datasetSize 13851
preds: tensor([11,  5, 10], device='cuda:0')

labels: tensor([11,  5, 10], device='cuda:0')

Loss = 0.08457352679184856

Acc = 0.9755252328351743

Reached first for loop: val
phase: val runningLoss: 328.88798810483706 datasetSize 3534
preds: tensor([ 8, 12], device='cuda:0')

labels: tensor([ 8, 12], device='cuda:0')

Loss = 0.09306394683215537

Acc = 0.9739671760045275

Epoch 4/9
----------
Reached first for loop: train
phase: train runningLoss: 977.4499700412161 datasetSize 13851
preds: tensor([11,  8, 11], device='cuda:0')

labels: tensor([11,  8, 11], device='cuda:0')

Loss = 0.07056890982898102

Acc = 0.9783409140134286

Reached first for loop: val
phase: val runningLoss: 562.686475788536 datasetSize 3534
preds: tensor([13, 11], device='cuda:0')

labels: tensor([13, 11], device='cuda:0')

Loss = 0.1592208477047357

Acc = 0.9533106960950765

Epoch 5/9
```

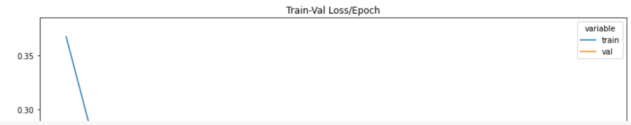
```
----------
Reached first for loop: train
phase: train runningLoss: 714.993083237001 datasetSize 13851
preds: tensor([11,  2,  0], device='cuda:0')

labels: tensor([11,  2,  0], device='cuda:0')

Loss = 0.051620322232113276

Acc = 0.9861381849685943

Reached first for loop: val
phase: val runningLoss: 130.49684724118833 datasetSize 3534
preds: tensor([11,  7], device='cuda:0')

labels: tensor([11,  7], device='cuda:0')

Loss = 0.036926102784716565

Acc = 0.9895302773061687

Epoch 6/9
----------
Reached first for loop: train
phase: train runningLoss: 661.2986311192799 datasetSize 13851
preds: tensor([ 7,  3, 13], device='cuda:0')

labels: tensor([ 7,  3, 13], device='cuda:0')

Loss = 0.04774374638071474

Acc = 0.9870767453613457

Reached first for loop: val
phase: val runningLoss: 139.47588862162047 datasetSize 3534
preds: tensor([11,  5], device='cuda:0')

labels: tensor([11,  5], device='cuda:0')

Loss = 0.03946686152281281

Acc = 0.9886813808715337

Epoch 7/9
----------
Reached first for loop: train
phase: train runningLoss: 443.01798428243274 datasetSize 13851
preds: tensor([10, 11,  1], device='cuda:0')

labels: tensor([10, 11,  1], device='cuda:0')

Loss = 0.03198454871723578

Acc = 0.99191394123168

Reached first for loop: val
phase: val runningLoss: 67.63638711437034 datasetSize 3534
preds: tensor([1, 0], device='cuda:0')
```

```
preds: tensor([1, 0], device='cuda:0')

labels: tensor([1, 0], device='cuda:0')

Loss = 0.019138762624326636

Acc = 0.9946236559139785

Epoch 8/9
----------
Reached first for loop: train
phase: train runningLoss: 230.22735342427404 datasetSize 13851
preds: tensor([ 1, 13,  7], device='cuda:0')

labels: tensor([ 1, 13,  7], device='cuda:0')

Loss = 0.01662171348092369

Acc = 0.9955959858493971

Reached first for loop: val
phase: val runningLoss: 48.5522757457926 datasetSize 3534
preds: tensor([5, 2], device='cuda:0')

labels: tensor([5, 2], device='cuda:0')

Loss = 0.013738617924672495
```

```
import pandas as pd
train_val_acc_df = pd.DataFrame.from_dict(accuracy_stats).reset_index().melt(id_vars=['index'
train_val_loss_df = pd.DataFrame.from_dict(loss_stats).reset_index().melt(id_vars=['index']).

# Plot line charts
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(30,10))
sns.lineplot(data=train_val_acc_df, x = "epochs", y="value", hue="variable",  ax=axes[0]).set
sns.lineplot(data=train_val_loss_df, x = "epochs", y="value", hue="variable", ax=axes[1]).set
```

```
Text(0.5, 1.0, 'Train-Val Loss/Epoch')
```






```
y_pred_list = []
y_true_list = []
phase='val'
with torch.no_grad():
    for inputs, labels in tqdm(dataloaders[phase]):
        inputs = inputs.to(device)
        labels = labels.to(device)
        # y_test_pred = model_ft(inputs)
        outputs = model_ft(inputs)
        # outputs = torch.log_softmax(outputs, dim=1)
        _, preds = torch.max(outputs, dim=1)
        y_pred_list.append(preds.cpu().numpy())
        y_true_list.append(labels.cpu().numpy())
```

100% 884/884 [00:12<00:00, 68.37it/s]

```
y_true_list[0]
```

```
array([13, 13,  5,  8])
```

```
y_pred_list = [i[0] for i in y_pred_list]
y_true_list = [i[0] for i in y_true_list]
```

```
len(y_true_list)
```

```
884
```

```
results = {
    "precision": precision_score(y_true_list, y_pred_list, average="weighted"),
    "recall": recall_score(y_true_list, y_pred_list,  average="weighted"),
    "f1": f1_score(y_true_list, y_pred_list, average="weighted")
}
print(results)
```

```
{'precision': 0.9955917764206818, 'recall': 0.995475113122172, 'f1': 0.9954780254158371
```

```
print(classification_report(y_true_list, y_pred_list))
```

```
              precision    recall  f1-score   support
```

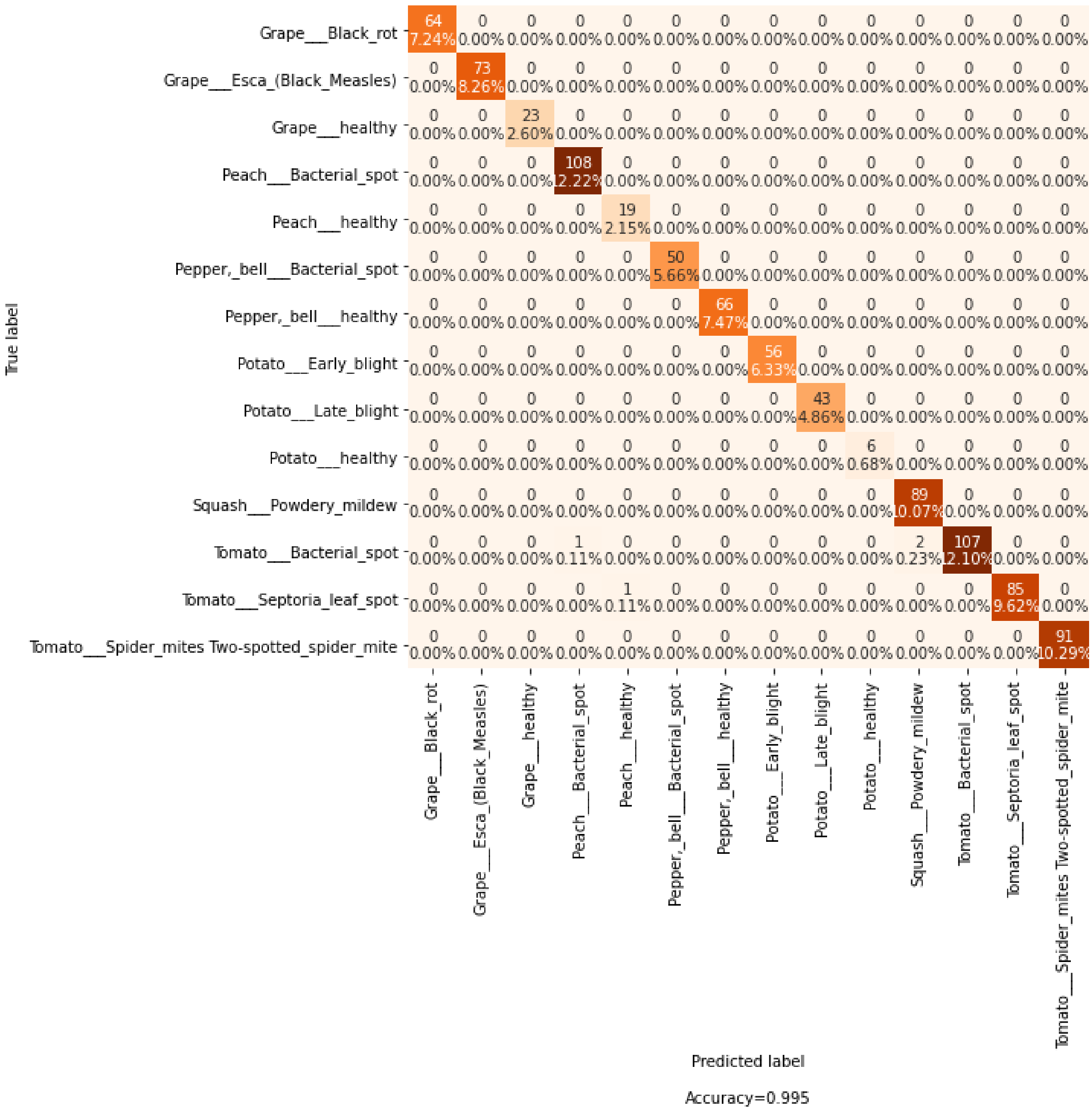
```
           0       1.00      1.00      1.00        64
           1       1.00      1.00      1.00        73
           2       1.00      1.00      1.00        23
           3       0.99      1.00      1.00       108
           4       0.95      1.00      0.97        19
           5       1.00      1.00      1.00        50
           6       1.00      1.00      1.00        66
           7       1.00      1.00      1.00        56
           8       1.00      1.00      1.00        43
           9       1.00      1.00      1.00         6
          10       0.98      1.00      0.99        89
          11       1.00      0.97      0.99       110
          12       1.00      0.99      0.99        86
          13       1.00      1.00      1.00        91

    accuracy                           1.00       884
   macro avg       0.99      1.00      1.00       884
weighted avg       1.00      1.00      1.00       884
```

```
cf=confusion_matrix(y_true_list, y_pred_list)
print(cf)
```

```
[[ 64   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0  73   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0  23   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0 108   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  19   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0  50   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  66   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  56   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  43   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   6   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  89   0   0   0]
 [  0   0   0   1   0   0   0   0   0   0   2 107   0   0]
 [  0   0   0   0   1   0   0   0   0   0   0   0  85   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  91]]
```

```
from cf_matrix import make_confusion_matrix
```

```


categories = class_names
# print(class_names)
make_confusion_matrix(cf,
                      categories=categories,
                      cmap='Oranges',
                      figsize=(8,8),
                      cbar=False,
                      sum_stats=True
                      )
```




```
import os
import torch
import torch.nn as nn
import torchvision
from torchvision import transforms
import json
import urllib
from PIL import Image
# Load the model
loaded_model = torch.load('model.pth')
loaded_model.eval()

# Load the labels
```