```
# import header files
%matplotlib inline
import torch
import torch.nn as nn
import torchvision
from functools import partial
from dataclasses import dataclass
from collections import OrderedDict
import glob
import os
import random
import tensorflow as tf
from tensorflow import keras
import numpy as np
import seaborn as sn
import pandas as pd
from matplotlib import pyplot as plt
from tqdm import tqdm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
import time
import copy
import tqdm
import torch
import random
from PIL import Image
import torch.optim as optim
from torchvision import models
import torch.nn.functional as F
import matplotlib.pyplot as plt
from torch.utils.data import TensorDataset,DataLoader
```

```
# load my google drive
def auth_gdrive():
  from google.colab import drive
  if os.path.exists('content/gdrive/My Drive'): return
  drive.mount('/content/gdrive')
def load_gdrive_dataset():
  loader_assets = 'MyPollen13K.zip'
  auth_gdrive()
```

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

```
Mounted at /content/gdrive
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_
```

```
# unzip  dataset
!unzip "/content/gdrive/MyDrive/MyPollen13K.zip"
```

```
Streaming output truncated to the last 5000 lines.
  inflating: MyPollen13K/train/3/20190404111856_OBJ_34_346_175.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_35_136_147.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_36_98_124.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_37_92_79.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_38_261_71.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_39_1097_44.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_3_330_866.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_40_317_606.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_41_160_362.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_5_1185_786.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_8_705_730.png
  inflating: MyPollen13K/train/3/20190404111856_OBJ_9_631_709.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_0_916_870.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_13_1126_429.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_15_978_405.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_16_1070_403.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_19_319_371.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_1_169_850.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_20_616_369.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_21_1191_359.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_24_706_354.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_25_1099_340.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_27_658_300.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_29_640_241.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_30_855_188.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_31_1127_177.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_32_528_174.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_33_341_167.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_34_681_134.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_36_886_102.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_38_1029_77.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_39_251_47.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_3_405_784.png
  inflating: MyPollen13K/train/3/20190404111901_OBJ_40_1223_542.png
```

```
inflating: MyPollen13K/train/3/20190404111901_OBJ_41_1164_520.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_42_683_459.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_43_1081_452.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_44_657_396.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_4_1069_751.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_5_653_688.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_6_1151_679.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_8_235_562.png
inflating: MyPollen13K/train/3/20190404111901_OBJ_9_850_545.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_0_1198_854.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_12_987_382.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_13_552_347.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_14_119_348.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_16_1000_271.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_17_698_252.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_18_550_205.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_19_85_165.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_1_447_771.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_20_73_112.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_2_416_736.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_6_1200_667.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_7_370_653.png
inflating: MyPollen13K/train/3/20190404111907_OBJ_8_300_555.png
```

```
# Count the number of samples in the training set and test set
# training set
train_class_1 = os.listdir("/content/MyPollen13K/train/1/")
train_class_1_samples = len(train_class_1)
print("The number of samples in the train_class_1 is:", train_class_1_samples)
train_class_2 = os.listdir("/content/MyPollen13K/train/2/")
train_class_2_samples = len(train_class_2)
print("The number of samples in the train_class_2 is:", train_class_2_samples)
train_class_3 = os.listdir("/content/MyPollen13K/train/3/")
train_class_3_samples = len(train_class_3)
print("The number of samples in the train_class_3 is:", train_class_3_samples)
train_class_4 = os.listdir("/content/MyPollen13K/train/4/")
train_class_4_samples = len(train_class_4)
print("The number of samples in the train_class_4 is:", train_class_4_samples)
number_trainingset = len(train_class_1+train_class_2+train_class_3+train_class_4)
print("The number of samples in the training set is:", number_trainingset)
# test set
test_class_1 = os.listdir("/content/MyPollen13K/test/1/")
test_class_1_samples = len(test_class_1)
print("The number of samples in the test_class_1 is:", test_class_1_samples)
test_class_2 = os.listdir("/content/MyPollen13K/test/2/")
test_class_2_samples = len(test_class_2)
print("The number of samples in the test_class_2 is:", test_class_2_samples)
test_class_3 = os.listdir("/content/MyPollen13K/test/3/")
test_class_3_samples = len(test_class_3)
print("The number of samples in the test_class_3 is:", test_class_3_samples)
test_class_4 = os.listdir("/content/MyPollen13K/test/4/")
test_class_4_samples = len(test_class_4)
print("The number of samples in the test_class_4 is:", test_class_4_samples)
number_testset = len(test_class_1+test_class_2+test_class_3+test_class_4)
print("The number of samples in the test set is:", number_testset)
```

```
The number of samples in the train_class_1 is: 1566
The number of samples in the train_class_2 is: 773
The number of samples in the train_class_3 is: 8216
The number of samples in the train_class_4 is: 724
The number of samples in the training set is: 11279
The number of samples in the test_class_1 is: 277
The number of samples in the test_class_2 is: 136
The number of samples in the test_class_3 is: 1450
The number of samples in the test_class_4 is: 128
The number of samples in the test set is: 1991
```

```
# define transforms
train_transforms = torchvision.transforms.Compose([torchvision.transforms.RandomRotation(30),
                                       torchvision.transforms.Resize((84, 84)),
                                       torchvision.transforms.RandomHorizontalFlip(),
                                       torchvision.transforms.ToTensor(),
                                       torchvision.transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])
```

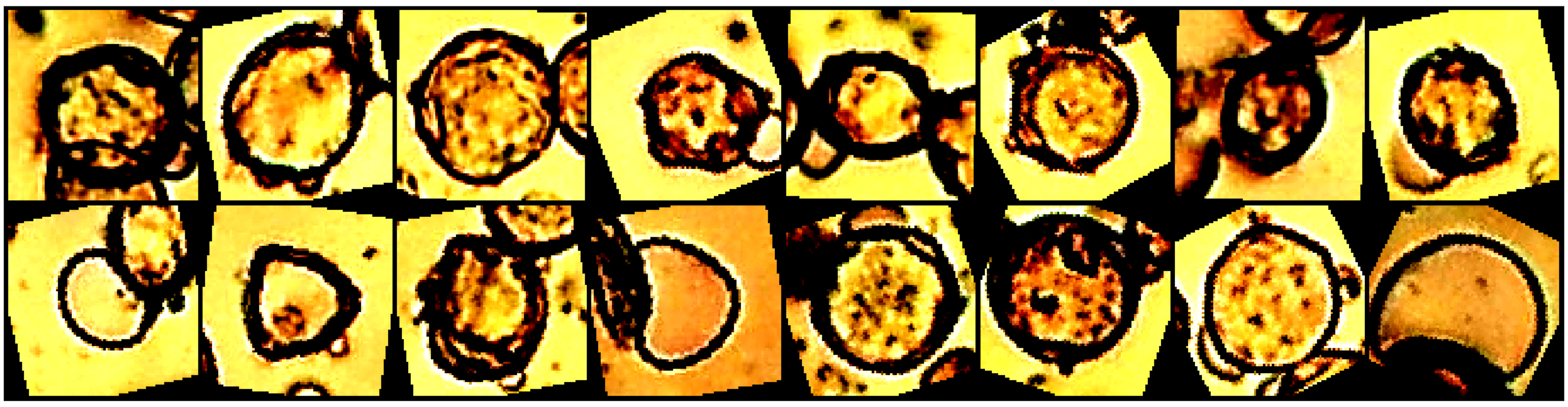
```
# get data
train_data = torchvision.datasets.ImageFolder("/content/MyPollen13K/train/", transform=train_transforms)
test_data = torchvision.datasets.ImageFolder("/content/MyPollen13K/test/", transform=train_transforms)
```

```
# data loader
trainloader = torch.utils.data.DataLoader(train_data, batch_size=16, shuffle=True, num_workers=1, pin_memory=True)
testloader = torch.utils.data.DataLoader(test_data, batch_size=16, shuffle=True, num_workers=1, pin_memory=True)
```

```
# Create a list of our detection classes
classes = ["1", "2", "3", "4"]
# plot random a batch images
from torchvision.utils import make_grid
def show_batch(dl, classes):
  for data, labels in dl:
```

```
    fig, ax = plt.subplots(figsize=(32, 16))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(data[:32], nrow=8).squeeze().permute(1, 2, 0).clamp(0,1))
    print('Labels: ', list(map(lambda l: classes[l], labels)))
    break
show_batch(trainloader, classes)
```

```
Labels:  ['3', '3', '3', '3', '3', '3', '3', '3', '4', '2', '3', '4', '3', '1', '3', '4']
```



```
# define the model
class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        residual = x
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(residual)
        out = F.relu(out)
        return out


class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion*planes, kernel_size=1, bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion*planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        residual = x
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(residual)
        out = F.relu(out)
        return out


class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=4):
```

```
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(2048*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out


def ResNet18():
    return ResNet(BasicBlock, [2,2,2,2])

def ResNet34():
    return ResNet(BasicBlock, [3,4,6,3])

def ResNet50():
    return ResNet(Bottleneck, [3,4,6,3])

def ResNet101():
    return ResNet(Bottleneck, [3,4,23,3])

def ResNet152():
    return ResNet(Bottleneck, [3,8,36,3])
```

```
# print the model
import math
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = ResNet18()
model.to(device)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (shortcut): Sequential()
    )
  )
```

```
    (layer3): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (shortcut): Sequential()
      )
    )
    (layer4): Sequential(
```

```
# print summary of the model
from torchvision import models
from torchsummary import summary
summary(model, (3, 84, 84))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 84, 84]           1,728
       BatchNorm2d-2           [-1, 64, 84, 84]             128
            Conv2d-3           [-1, 64, 84, 84]          36,864
       BatchNorm2d-4           [-1, 64, 84, 84]             128
            Conv2d-5           [-1, 64, 84, 84]          36,864
       BatchNorm2d-6           [-1, 64, 84, 84]             128
        BasicBlock-7           [-1, 64, 84, 84]               0
            Conv2d-8           [-1, 64, 84, 84]          36,864
       BatchNorm2d-9           [-1, 64, 84, 84]             128
           Conv2d-10           [-1, 64, 84, 84]          36,864
      BatchNorm2d-11           [-1, 64, 84, 84]             128
       BasicBlock-12           [-1, 64, 84, 84]               0
           Conv2d-13          [-1, 128, 42, 42]          73,728
      BatchNorm2d-14          [-1, 128, 42, 42]             256
           Conv2d-15          [-1, 128, 42, 42]         147,456
      BatchNorm2d-16          [-1, 128, 42, 42]             256
           Conv2d-17          [-1, 128, 42, 42]           8,192
      BatchNorm2d-18          [-1, 128, 42, 42]             256
       BasicBlock-19          [-1, 128, 42, 42]               0
           Conv2d-20          [-1, 128, 42, 42]         147,456
      BatchNorm2d-21          [-1, 128, 42, 42]             256
           Conv2d-22          [-1, 128, 42, 42]         147,456
      BatchNorm2d-23          [-1, 128, 42, 42]             256
       BasicBlock-24          [-1, 128, 42, 42]               0
           Conv2d-25          [-1, 256, 21, 21]         294,912
      BatchNorm2d-26          [-1, 256, 21, 21]             512
           Conv2d-27          [-1, 256, 21, 21]         589,824
      BatchNorm2d-28          [-1, 256, 21, 21]             512
           Conv2d-29          [-1, 256, 21, 21]          32,768
      BatchNorm2d-30          [-1, 256, 21, 21]             512
       BasicBlock-31          [-1, 256, 21, 21]               0
           Conv2d-32          [-1, 256, 21, 21]         589,824
      BatchNorm2d-33          [-1, 256, 21, 21]             512
           Conv2d-34          [-1, 256, 21, 21]         589,824
      BatchNorm2d-35          [-1, 256, 21, 21]             512
       BasicBlock-36          [-1, 256, 21, 21]               0
           Conv2d-37          [-1, 512, 11, 11]       1,179,648
      BatchNorm2d-38          [-1, 512, 11, 11]           1,024
           Conv2d-39          [-1, 512, 11, 11]       2,359,296
      BatchNorm2d-40          [-1, 512, 11, 11]           1,024
           Conv2d-41          [-1, 512, 11, 11]         131,072
      BatchNorm2d-42          [-1, 512, 11, 11]           1,024
       BasicBlock-43          [-1, 512, 11, 11]               0
           Conv2d-44          [-1, 512, 11, 11]       2,359,296
      BatchNorm2d-45          [-1, 512, 11, 11]           1,024
           Conv2d-46          [-1, 512, 11, 11]       2,359,296
      BatchNorm2d-47          [-1, 512, 11, 11]           1,024
       BasicBlock-48          [-1, 512, 11, 11]               0
           Linear-49                    [-1, 4]           8,196
================================================================
Total params: 11,177,028
Trainable params: 11,177,028
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
```

```
# loss function to be used
criterion = torch.nn.CrossEntropyLoss()
# optimizer to be used
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=5e-3, momentum=0.9, weight_decay=5e-4)
```

```
# training process
from torch.utils.tensorboard import SummaryWriter
train_losses = 0.0
train_accuracy = 0
epochs = 50
for epoch in range(epochs):  # loop over the dataset multiple times
    print('Epoch-{0}:'.format(epoch + 1, optimizer.param_groups[0]['lr']))
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data # get the inputs; data is a list of [inputs, labels]
        inputs, labels = inputs.cuda(), labels.cuda() # for using data in GPU
        optimizer.zero_grad() # zero the parameter gradients
        outputs = model(inputs) # forward
        loss = criterion(outputs, labels) # calculate loss
        loss.backward() # backward loss
        optimizer.step() # optimize gradients
        train_losses += loss.item() # save loss
        _, preds = torch.max(outputs, 1) # save prediction
        train_accuracy += torch.sum(preds == labels.data) # save train_accuracy
        if i % 1000 == 999:    # every 1000 mini-batches...
            steps = epoch * len(trainloader) + i # calculate steps
            batch = i*batch_size # calculate batch
            print("Training loss {:.5} Training Accuracy {:.5} Steps: {}".format(train_losses / batch, train_accuracy/batch, steps))
            # Save train_accuracy and loss to Tensorboard
            writer.add_scalar('Training loss by steps', train_losses / batch, steps)
            writer.add_scalar('Training accuracy by steps', train_accuracy / batch, steps)
    print("Training Accuracy: {}/{} ({:.5} %) Training Loss: {:.5}".format(train_accuracy, len(trainloader), 100. * train_accuracy / len(tra
    train_losses = 0.0
    train_accuracy = 0
print('Train is finished...')
```

```
Epoch-1:
Training Accuracy: 4534/542 (52.344 %) Training Loss: 0.093382
Epoch-2:
Training Accuracy: 6452/542 (74.486 %) Training Loss: 0.04886
Epoch-3:
Training Accuracy: 7234/542 (83.514 %) Training Loss: 0.031848
Epoch-4:
Training Accuracy: 7594/542 (87.67 %) Training Loss: 0.023548
Epoch-5:
Training Accuracy: 7895/542 (91.145 %) Training Loss: 0.016865
Epoch-6:
Training Accuracy: 8030/542 (92.704 %) Training Loss: 0.013711
Epoch-7:
Training Accuracy: 8169/542 (94.308 %) Training Loss: 0.010641
Epoch-8:
Training Accuracy: 8327/542 (96.133 %) Training Loss: 0.0073051
Epoch-9:
Training Accuracy: 8359/542 (96.502 %) Training Loss: 0.0064149
Epoch-10:
Training Accuracy: 8387/542 (96.825 %) Training Loss: 0.0060478
Epoch-11:
Training Accuracy: 8416/542 (97.16 %) Training Loss: 0.0054738
Epoch-12:
Training Accuracy: 8477/542 (97.864 %) Training Loss: 0.0041742
Epoch-13:
Training Accuracy: 8498/542 (98.107 %) Training Loss: 0.003767
Epoch-14:
Training Accuracy: 8507/542 (98.211 %) Training Loss: 0.0033845
Epoch-15:
Training Accuracy: 8517/542 (98.326 %) Training Loss: 0.0031471
Epoch-16:
Training Accuracy: 8519/542 (98.349 %) Training Loss: 0.0029143
Epoch-17:
Training Accuracy: 8570/542 (98.938 %) Training Loss: 0.0024028
Epoch-18:
Training Accuracy: 8576/542 (99.007 %) Training Loss: 0.0019961
Epoch-19:
Training Accuracy: 8561/542 (98.834 %) Training Loss: 0.0023486
Epoch-20:
Training Accuracy: 8591/542 (99.18 %) Training Loss: 0.0016557
Epoch-21:
Training Accuracy: 8587/542 (99.134 %) Training Loss: 0.0018328
Epoch-22:
Training Accuracy: 8601/542 (99.296 %) Training Loss: 0.0015853
Epoch-23:
Training Accuracy: 8575/542 (98.996 %) Training Loss: 0.0020609
Epoch-24:
Training Accuracy: 8579/542 (99.042 %) Training Loss: 0.0021052
Epoch-25:
Training Accuracy: 8592/542 (99.192 %) Training Loss: 0.001678
Epoch-26:
Training Accuracy: 8583/542 (99.088 %) Training Loss: 0.0018683
Epoch-27:
Training Accuracy: 8617/542 (99.48 %) Training Loss: 0.001128
Epoch-28:
Training Accuracy: 8587/542 (99.134 %) Training Loss: 0.0017009
Epoch-29:
Training Accuracy: 8633/542 (99.665 %) Training Loss: 0.00077358
```

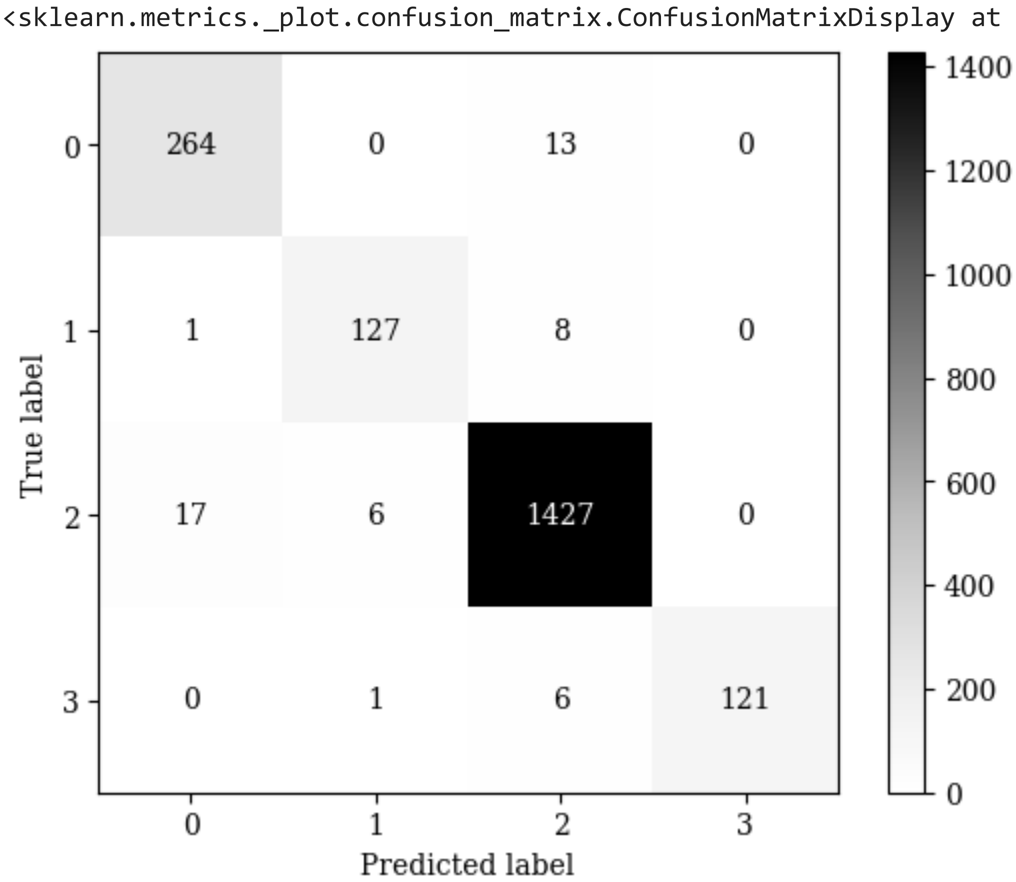
Training Accuracy: 8855/542 (99.885 %) Training Loss: 0.00077958

```
# test proess
from torch.utils.tensorboard import SummaryWriter
test_losses = 0.0
test_accuracy = 0
epochs = 50
for epoch in range(epochs):  # loop over the dataset multiple times
    print('Epoch-{0}:'.format(epoch + 1, optimizer.param_groups[0]['lr']))
    for i, data in enumerate(testloader, 0):
        inputs, labels = data # get the inputs; data is a list of [inputs, labels]
        inputs, labels = inputs.cuda(), labels.cuda() # for using data in GPU
        optimizer.zero_grad() # zero the parameter gradients
        outputs = model(inputs) # forward
        loss = criterion(outputs, labels) # calculate loss
        loss.backward() # backward loss
        optimizer.step() # optimize gradients
        test_losses += loss.item() # save loss
        _, preds = torch.max(outputs, 1) # save prediction
        test_accuracy += torch.sum(preds == labels.data) # save test_accuracy
        if i % 1000 == 999:    # every 1000 mini-batches...
            steps = epoch * len(testloader) + i # calculate steps
            batch = i*batch_size # calculate batch
            print("Test loss {:.5} Test Accuracy {:.5} Steps: {}".format(test_losses / batch, test_accuracy/batch, steps))
            # Save test_accuracy and loss to Tensorboard
            writer.add_scalar('Test loss by steps', test_losses / batch, steps)
            writer.add_scalar('Test accuracy by steps', test_accuracy / batch, steps)
    print("Test Accuracy: {}/{} ({:.5} %) Test Loss: {:.5}".format(test_accuracy, len(testloader), 100. * test_accuracy / len(testloader.dat
    test_losses = 0.0
    test_accuracy = 0
print('Test is Finished...')
```

```
Epoch-1:
Test Accuracy: 1743/127 (46.415 %) Test Loss: 0.13001
Epoch-2:
Test Accuracy: 1850/127 (51.72 %) Test Loss: 0.0919012
Epoch-3:
Test Accuracy: 1863/127 (62.365 %) Test Loss: 0.0815945
Epoch-4:
Test Accuracy: 1913/127 (74.844 %) Test Loss: 0.07103
Epoch-5:
Test Accuracy: 1926/127 (85.488 %) Test Loss: 0.06094844
Epoch-6:
Test Accuracy: 1893/127 (93.852 %) Test Loss: 0.01196
Epoch-7:
Test Accuracy: 1956/127 (96.976 %) Test Loss: 0.0091549
Epoch-8:
Test Accuracy: 1842/127 (91.324 %) Test Loss: 0.017049
Epoch-9:
Test Accuracy: 1924/127 (95.389 %) Test Loss: 0.0081645
Epoch-10:
Test Accuracy: 1947/127 (96.53 %) Test Loss: 0.0073562
Epoch-11:
Test Accuracy: 1898/127 (94.1 %) Test Loss: 0.013155
Epoch-12:
Test Accuracy: 1919/127 (95.141 %) Test Loss: 0.011162
Epoch-13:
Test Accuracy: 1859/127 (92.167 %) Test Loss: 0.015105
Epoch-14:
Test Accuracy: 1926/127 (95.488 %) Test Loss: 0.0094164
Epoch-15:
Test Accuracy: 1951/127 (96.728 %) Test Loss: 0.0070101
Epoch-16:
Test Accuracy: 1973/127 (97.819 %) Test Loss: 0.0043344
Epoch-17:
Test Accuracy: 1982/127 (98.265 %) Test Loss: 0.0036788
Epoch-18:
Test Accuracy: 1957/127 (97.025 %) Test Loss: 0.0061826
Epoch-19:
Test Accuracy: 1990/127 (98.661 %) Test Loss: 0.0034277
Epoch-20:
Test Accuracy: 1984/127 (98.364 %) Test Loss: 0.0038971
Epoch-21:
Test Accuracy: 1921/127 (95.24 %) Test Loss: 0.011374
Epoch-22:
Test Accuracy: 1945/127 (96.43 %) Test Loss: 0.0077045
Epoch-23:
Test Accuracy: 1980/127 (98.166 %) Test Loss: 0.0063771
Epoch-24:
Test Accuracy: 1957/127 (97.025 %) Test Loss: 0.0070434
Epoch-25:
Test Accuracy: 1988/127 (98.562 %) Test Loss: 0.0040153
Epoch-26:
Test Accuracy: 1910/127 (94.695 %) Test Loss: 0.010725
Epoch-27:
Test Accuracy: 1996/127 (98.959 %) Test Loss: 0.0026239
Epoch-28:
Test Accuracy: 2001/127 (99.207 %) Test Loss: 0.0027295
Epoch-29:
Test Accuracy: 1928/127 (95.588 %) Test Loss: 0.010983
```

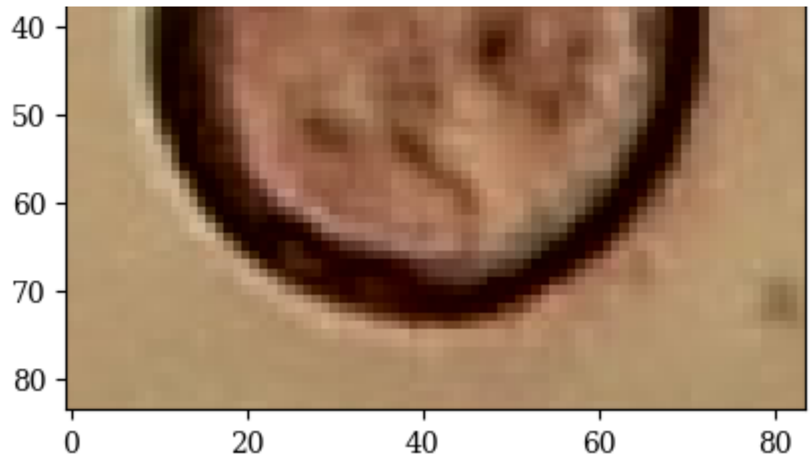
```
# import Times New Roman font
import matplotlib.font_manager
!wget https://github.com/trishume/OpenTuringCompiler/blob/master/stdlib-sfml/fonts/Times%20New%20Roman.ttf -P /usr/local/lib/python3.6/dist-pa
import matplotlib.pyplot as plt
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = ['Times New Roman'] + plt.rcParams['font.serif']
# test confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import ConfusionMatrixDisplay
import seaborn as sn
import pandas as pd
y_pred = []
y_true = []
# iterate over test data
for inputs, labels in testloader:
        inputs, labels = inputs.cuda(), labels.cuda()
        output = model(inputs) # Feed Network
        output = (torch.max(torch.exp(output), 1)[1]).data.cpu().numpy()
        y_pred.extend(output) # Save Prediction
        labels = labels.data.cpu().numpy()
        y_true.extend(labels) # Save Truth
cm = confusion_matrix(y_true, y_pred)
cm_display = ConfusionMatrixDisplay(cm)
cm_display.plot(cmap=plt.cm.Greys)
```

```
--2024-01-11 12:10:10--  https://github.com/trishume/OpenTuringCompiler/blob/master/stdlib-sfml/fonts/Times%20New%20
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5705 (5.6K) [text/plain]
Saving to: '/usr/local/lib/python3.6/dist-packages/matplotlib/mpl-data/fonts/ttf/Times New Roman.ttf.3'

Times New Roman.ttf 100%[===================>]   5.57K  --.-KB/s    in 0s

2024-01-11 12:10:10 (73.1 MB/s) - '/usr/local/lib/python3.6/dist-packages/matplotlib/mpl-data/fonts/ttf/Times New Ro

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x78ba28f7c9a0>
```




```
import tensorflow
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
image = load_img('/content/MyPollen13K/train/1/20190402165648_OBJ_0_1099_759.png')
data = img_to_array(image)
samples = np.expand_dims(data, 0)
print('An image of class1 (Corylus avellana_well developed):')
plt.imshow(image)
plt.show()
```

```
An image of class1 (Corylus avellana_well developed):
```




```
from torchvision import models, transforms, utils
transform = transforms.Compose([
    transforms.Resize((84, 84)),
    transforms.ToTensor(),
    transforms.Normalize(mean=0., std=1.)
])
# we will save the conv layer weights in this list
model_weights =[]
# we will save the 49 conv layers in this list
conv_layers = []
# get all the model children as list
model_children = list(model.children())
# counter to keep count of the conv layers
counter = 0
# append all the conv layers and their respective wights to the list
for i in range(len(model_children)):
    if type(model_children[i]) == nn.Conv2d:
        counter+=1
        model_weights.append(model_children[i].weight)
        conv_layers.append(model_children[i])
    elif type(model_children[i]) == nn.Sequential:
        for j in range(len(model_children[i])):
            for child in model_children[i][j].children():
                if type(child) == nn.Conv2d:
                    counter+=1
                    model_weights.append(child.weight)
                    conv_layers.append(child)
print(f"Total convolution layers: {counter}")
print("conv_layers")
```

```
Total convolution layers: 17
conv_layers
```

```
from torch.autograd import Variable
import matplotlib.pyplot as plt
import scipy.misc
from PIL import Image
import json
%matplotlib inline
image = transform(image)
print(f"Image shape before: {image.shape}")
image = image.unsqueeze(0)
print(f"Image shape after: {image.shape}")
image = image.to(device)
```

```
Image shape before: torch.Size([3, 84, 84])
Image shape after: torch.Size([1, 3, 84, 84])
```

```
outputs = []
names = []
for layer in conv_layers[0:]:
    image = layer(image)
    outputs.append(image)
    names.append(str(layer))
print(len(outputs))
# print feature_maps
for feature_map in outputs:
    print(feature_map.shape)
```

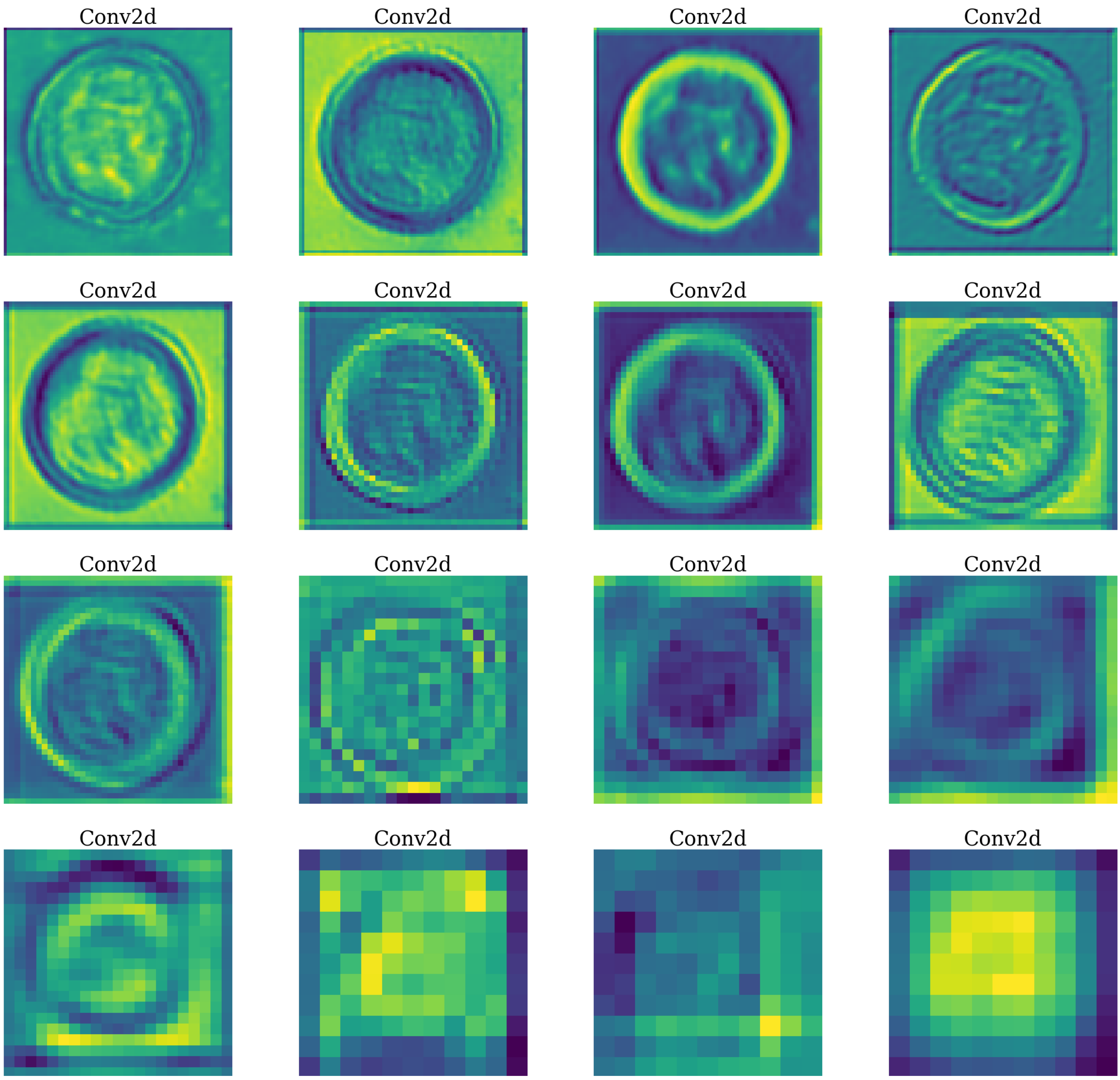
```
17
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 512, 11, 11])
```

```
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
```

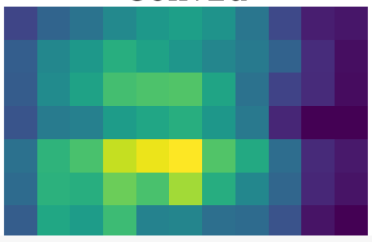
```
processed = []
for feature_map in outputs:
    feature_map = feature_map.squeeze(0)
    gray_scale = torch.sum(feature_map,0)
    gray_scale = gray_scale / feature_map.shape[0]
    processed.append(gray_scale.data.cpu().numpy())
for fm in processed:
    print(fm.shape)
```

```
(84, 84)
(84, 84)
(84, 84)
(84, 84)
(84, 84)
(42, 42)
(42, 42)
(42, 42)
(42, 42)
(21, 21)
(21, 21)
(21, 21)
(21, 21)
(11, 11)
(11, 11)
(11, 11)
(11, 11)
```

```
# print Corylus avellana_well developed feature maps
fig = plt.figure(figsize=(30, 50))
for i in range(len(processed)):
    a = fig.add_subplot(7, 4, i+1)
    imgplot = plt.imshow(processed[i])
    a.axis("off")
    a.set_title(names[i].split('(')[0], fontsize=30)
```

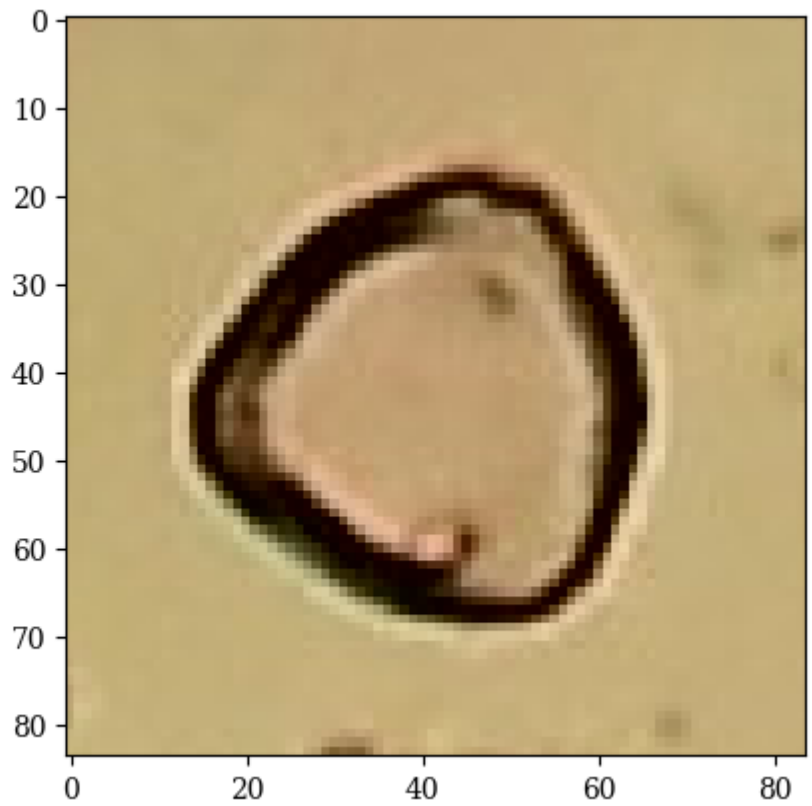


Conv2d



```
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
image = load_img('/content/MyPollen13K/train/2/20190404110723_OBJ_42_791_49.png')
data = img_to_array(image)
samples = np.expand_dims(data, 0)
print('An image of class2 (Corylus avellana_anomalous):')
plt.imshow(image)
plt.show()
```

```
An image of class2 (Corylus avellana_anomalous):
```




```
from torchvision import models, transforms, utils
transform = transforms.Compose([
    transforms.Resize((84, 84)),
    transforms.ToTensor(),
    transforms.Normalize(mean=0., std=1.)
])
```

```
# we will save the conv layer weights in this list
model_weights =[]
#we will save the 49 conv layers in this list
conv_layers = []
# get all the model children as list
model_children = list(model.children())
# counter to keep count of the conv layers
counter = 0
# append all the conv layers and their respective wights to the list
for i in range(len(model_children)):
    if type(model_children[i]) == nn.Conv2d:
        counter+=1
        model_weights.append(model_children[i].weight)
        conv_layers.append(model_children[i])
    elif type(model_children[i]) == nn.Sequential:
        for j in range(len(model_children[i])):
            for child in model_children[i][j].children():
                if type(child) == nn.Conv2d:
                    counter+=1
                    model_weights.append(child.weight)
                    conv_layers.append(child)
print(f"Total convolution layers: {counter}")
print("conv_layers")
```

```
Total convolution layers: 17
conv_layers
```

```
from torch.autograd import Variable
import matplotlib.pyplot as plt
import scipy.misc
from PIL import Image
import json
%matplotlib inline
image = transform(image)
print(f"Image shape before: {image.shape}")
image = image.unsqueeze(0)
print(f"Image shape after: {image.shape}")
```

```
image = image.to(device)
```

```
Image shape before: torch.Size([3, 84, 84])
Image shape after: torch.Size([1, 3, 84, 84])
```

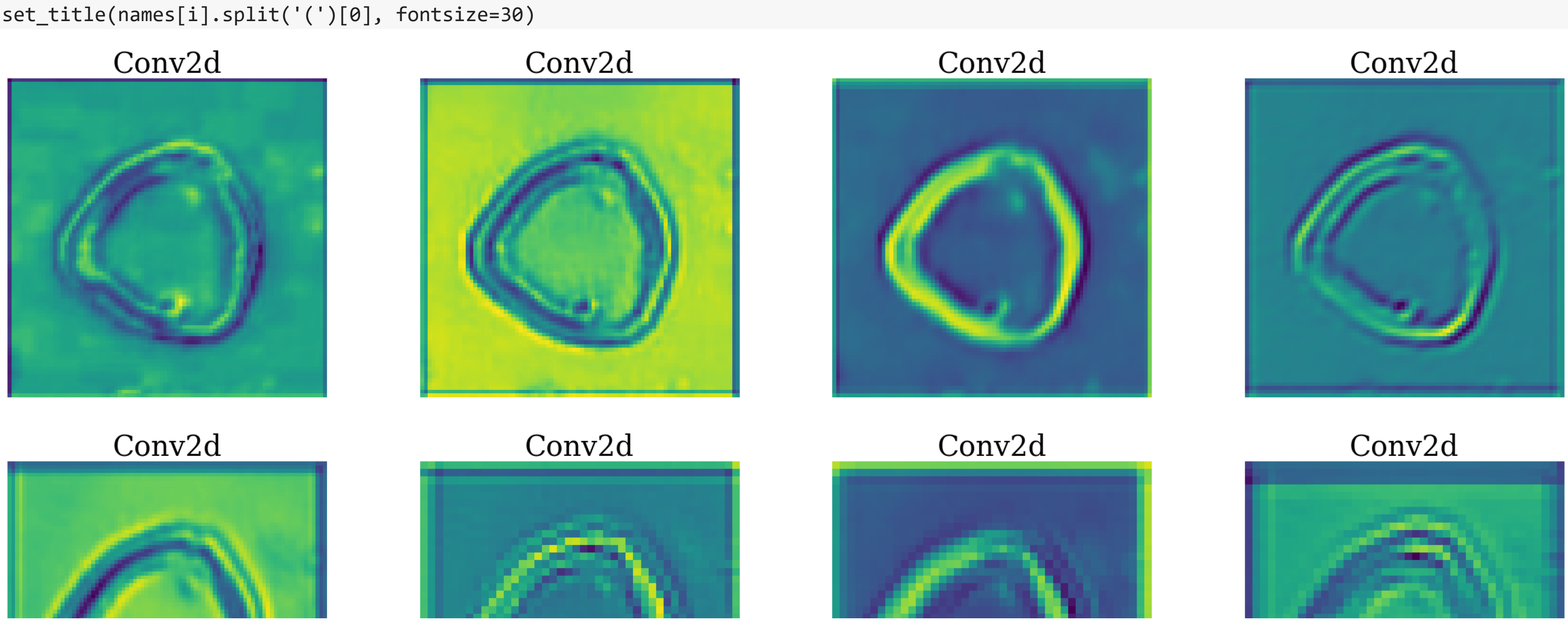
```
outputs = []
names = []
for layer in conv_layers[0:]:
    image = layer(image)
    outputs.append(image)
    names.append(str(layer))
print(len(outputs))
# print feature_maps
for feature_map in outputs:
    print(feature_map.shape)
```

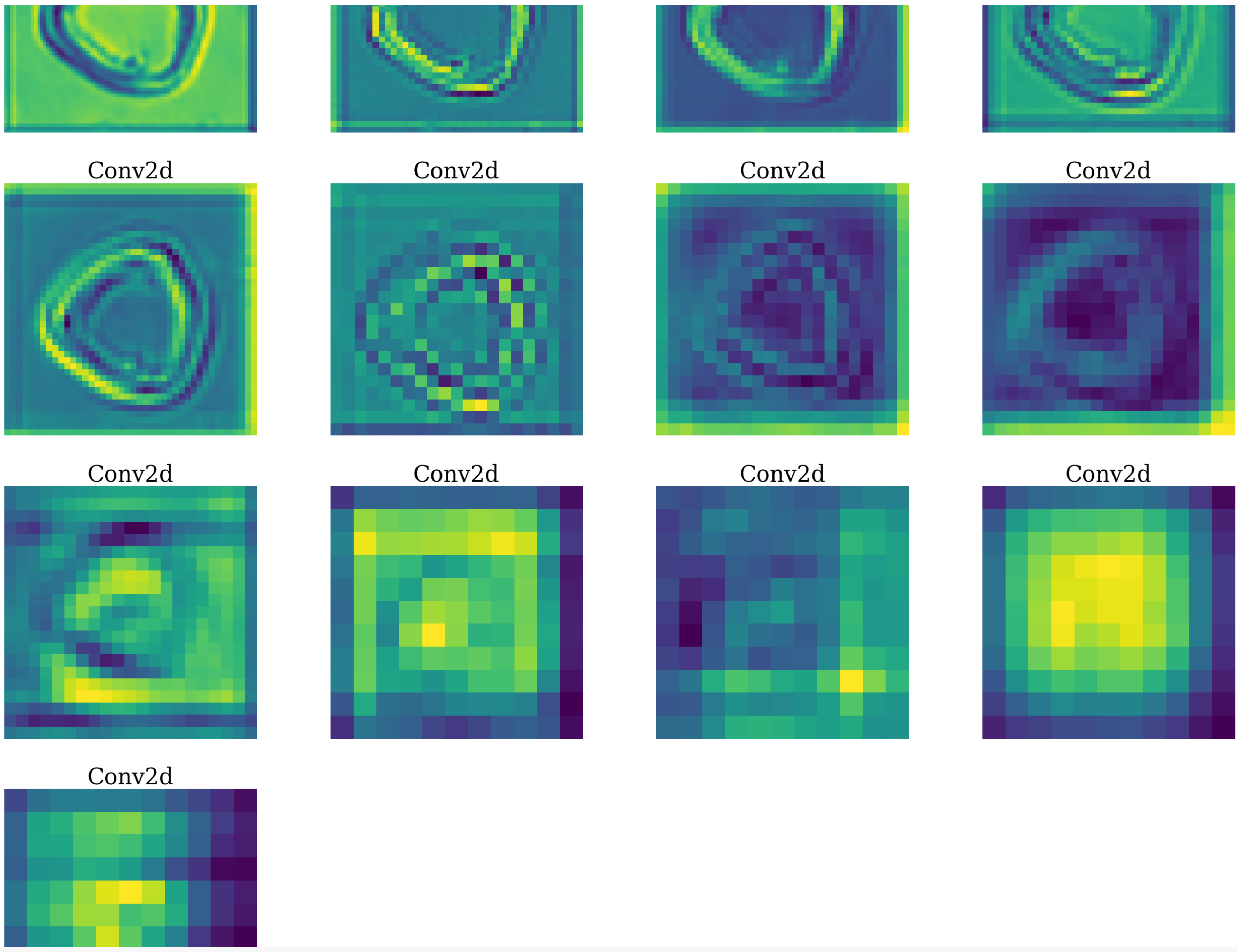
```
17
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
```

```
processed = []
for feature_map in outputs:
    feature_map = feature_map.squeeze(0)
    gray_scale = torch.sum(feature_map,0)
    gray_scale = gray_scale / feature_map.shape[0]
    processed.append(gray_scale.data.cpu().numpy())
for fm in processed:
    print(fm.shape)
```

```
(84, 84)
(84, 84)
(84, 84)
(84, 84)
(84, 84)
(42, 42)
(42, 42)
(42, 42)
(42, 42)
(21, 21)
(21, 21)
(21, 21)
(21, 21)
(11, 11)
(11, 11)
(11, 11)
(11, 11)
```

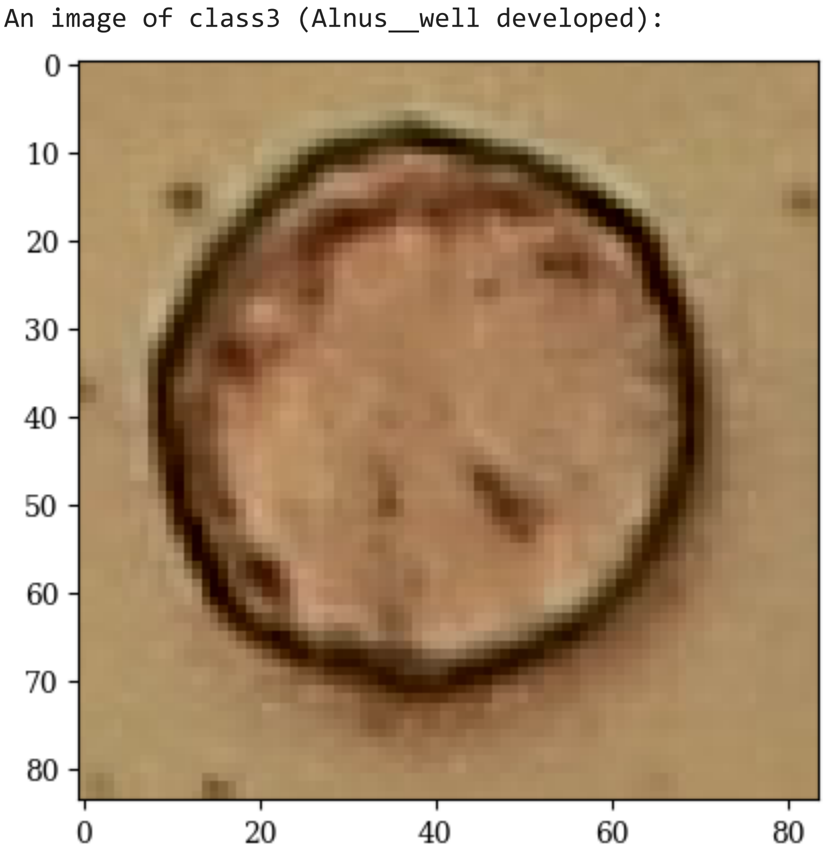
```
# print Corylus avellana_anomalous feature maps
fig = plt.figure(figsize=(30, 50))
for i in range(len(processed)):
    a = fig.add_subplot(7, 4, i+1)
    imgplot = plt.imshow(processed[i])
    a.axis("off")
    a.set_title(names[i].split('(')[0], fontsize=30)
```






```
import tensorflow
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
image = load_img('/content/MyPollen13K/train/3/20190404105005_OBJ_0_933_905.png')
data = img_to_array(image)
samples = np.expand_dims(data, 0)
print('An image of class3 (Alnus__well developed):')
plt.imshow(image)
plt.show()
```

```
An image of class3 (Alnus__well developed):
```




```
from torchvision import models, transforms, utils
transform = transforms.Compose([
    transforms.Resize((84, 84)),
    transforms.ToTensor(),
    transforms.Normalize(mean=0., std=1.)
])
```

```
# we will save the conv layer weights in this list
model_weights =[]
# we will save the 49 conv layers in this list
conv_layers = []
# get all the model children as list
model_children = list(model.children())
```

```
model_children = list(model.children())
# counter to keep count of the conv layers
counter = 0
# append all the conv layers and their respective wights to the list
for i in range(len(model_children)):
    if type(model_children[i]) == nn.Conv2d:
        counter+=1
        model_weights.append(model_children[i].weight)
        conv_layers.append(model_children[i])
    elif type(model_children[i]) == nn.Sequential:
        for j in range(len(model_children[i])):
            for child in model_children[i][j].children():
                if type(child) == nn.Conv2d:
                    counter+=1
                    model_weights.append(child.weight)
                    conv_layers.append(child)
print(f"Total convolution layers: {counter}")
print("conv_layers")
```

```
Total convolution layers: 17
conv_layers
```

```
from torch.autograd import Variable
import matplotlib.pyplot as plt
import scipy.misc
from PIL import Image
import json
%matplotlib inline
image = transform(image)
print(f"Image shape before: {image.shape}")
image = image.unsqueeze(0)
print(f"Image shape after: {image.shape}")
image = image.to(device)
```

```
Image shape before: torch.Size([3, 84, 84])
Image shape after: torch.Size([1, 3, 84, 84])
```

```
outputs = []
names = []
for layer in conv_layers[0:]:
    image = layer(image)
    outputs.append(image)
    names.append(str(layer))
print(len(outputs))
# print feature_maps
for feature_map in outputs:
    print(feature_map.shape)
```

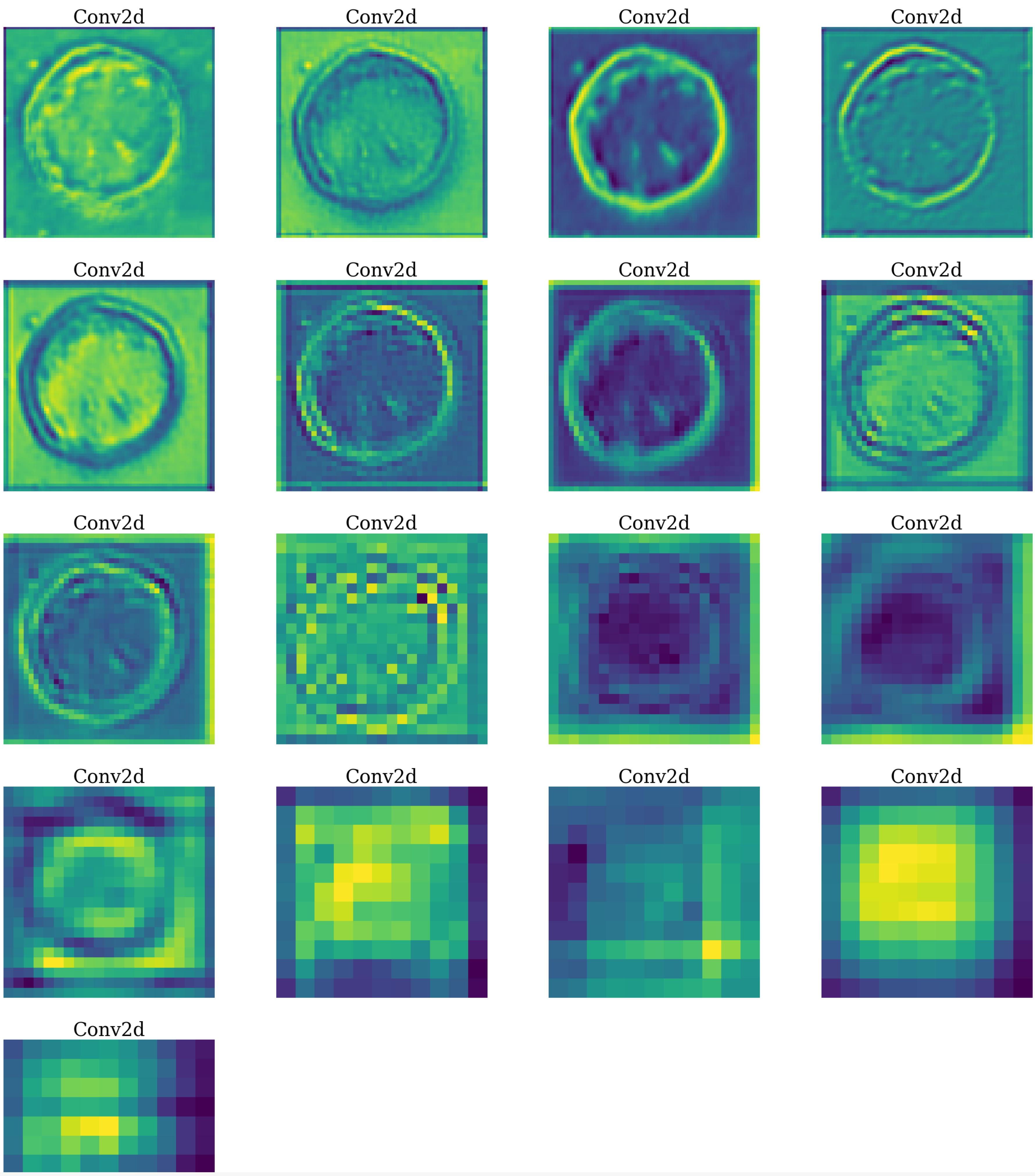
```
17
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 64, 84, 84])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 128, 42, 42])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 256, 21, 21])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
torch.Size([1, 512, 11, 11])
```

```
processed = []
for feature_map in outputs:
    feature_map = feature_map.squeeze(0)
    gray_scale = torch.sum(feature_map,0)
    gray_scale = gray_scale / feature_map.shape[0]
    processed.append(gray_scale.data.cpu().numpy())
for fm in processed:
    print(fm.shape)
```

```
(84, 84)
(84, 84)
(84, 84)
(84, 84)
(84, 84)
(42, 42)
(42, 42)
(42, 42)
(42, 42)
(21, 21)
(21, 21)
(21, 21)
(21, 21)
(11, 11)
```

```
(11, 11)
(11, 11)
(11, 11)
```

```
# print Alnus__well developed feature maps
fig = plt.figure(figsize=(30, 50))
for i in range(len(processed)):
    a = fig.add_subplot(7, 4, i+1)
    imgplot = plt.imshow(processed[i])
    a.axis("off")
    a.set_title(names[i].split('(')[0], fontsize=30)
```




```
!pip install git+https://github.com/jacobgil/pytorch-grad-cam.git
```

```
Collecting git+https://github.com/jacobgil/pytorch-grad-cam.git
  Cloning https://github.com/jacobgil/pytorch-grad-cam.git to /tmp/pip-req-build-fqzm17sl
  Running command git clone --filter=blob:none --quiet https://github.com/jacobgil/pytorch-grad-cam.git /tmp/pip-req
  Resolved https://github.com/jacobgil/pytorch-grad-cam.git to commit 09ac162e8f609eed02a8e35a370ef5bf30de19a1
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (1.23.5)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (9.4.0)
Requirement already satisfied: torch>=1.7.1 in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (2.0.1
Requirement already satisfied: torchvision>=0.8.2 in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8)
Collecting ttach (from grad-cam==1.4.8)
  Downloading ttach-0.0.3-py3-none-any.whl (9.8 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (4.66.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (4.8.
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (3.7.1)
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from grad-cam==1.4.8) (1.2.2
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.7.1->grad-cam==1.4
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch>=1.7.1->grad
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.7.1->grad-cam==1.4.8)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.7.1->grad-cam==1.4
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.7.1->grad-cam==1.4.8
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.7.1->grad-cam
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch>=1.7.1->g
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch>=1.7.1->gra
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision>=0.8.2->grad-ca
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->grad-ca
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->grad-cam==1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->grad-c
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->grad-c
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->grad-cam
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->grad-ca
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->gra
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->grad-cam=
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->grad-cam
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->g
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matpl
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.7.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->t
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision>=
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvi
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvi
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.7.1->gr
Building wheels for collected packages: grad-cam
  Building wheel for grad-cam (pyproject.toml) ... done
  Created wheel for grad-cam: filename=grad_cam-1.4.8-py3-none-any.whl size=37447 sha256=12500a394d4987b4b2e30811af1
  Stored in directory: /tmp/pip-ephem-wheel-cache-nukftv8t/wheels/23/11/66/71a38b0c29ba4ec5f62105a2145278613855bc9c9
Successfully built grad-cam
Installing collected packages: ttach, grad-cam
Successfully installed grad-cam-1.4.8 ttach-0.0.3
```

```
import copy
from pytorch_grad_cam import GradCAM, ScoreCAM, GradCAMPlusPlus, AblationCAM, XGradCAM, EigenCAM, FullGrad
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image
from torchvision.models import resnet18
import numpy as np
from PIL import Image
import torch
import torch.nn as nn
import torchvision
```

```
# Pick up layers for visualization
target_layers = [model.layer4[-1]]
```

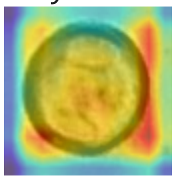
```
path1 = ('/content/MyPollen13K/train/1/20190402165648_OBJ_0_1099_759.png')
print('Corylus avellana_well developed:')
Image.open(path1).convert('RGB')
```

```
Corylus avellana_well developed:
```

```
rgb_img = Image.open(path1).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
# cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=False)
# cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=False)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
# plot Corylus avellana_well developed GradCAM
print('Corylus avellana_well developed GradCAM:')
Image.fromarray(visualization, 'RGB')
```

```
Corylus avellana_well developed GradCAM:
```



```
rgb_img = Image.open(path1).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
#cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
# cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=False)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
# plot Corylus avellana_well developed GradCAMPlusPlus
print('Corylus avellana_well developed GradCAMPlusPlus')
Image.fromarray(visualization, 'RGB')
```

```
Corylus avellana_well developed GradCAMPlusPlus
```

```
rgb_img = Image.open(path1).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
#cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
#cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=True)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
100%|██████████| 32/32 [00:01<00:00, 25.80it/s]
```

```
# plot Corylus avellana_well developed ScoreCAM
print('Corylus avellana_well developed ScoreCAM:')
Image.fromarray(visualization, 'RGB')
```

```
Corylus avellana_well developed ScoreCAM:
```

```
path2 = ('/content/MyPollen13K/train/2/20190404110723_OBJ_42_791_49.png')
print('Corylus avellana_anomalous:')
Image.open(path2).convert('RGB')
```

Corylus avellana_anomalous:

```
rgb_img = Image.open(path2).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!

# Construct the CAM object once, and then re-use it on many images:
cam1 = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
#cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
# cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=False)

# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...

# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None

# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam1 = cam1(input_tensor=input_tensor)

# In this example grayscale_cam1 has only one image in the batch:
grayscale_cam1 = grayscale_cam1[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam1, use_rgb=True)
```

```
# plot Corylus avellana_anomalous GradCAM
print('Corylus avellana_anomalous GradCAM:')
Image.fromarray(visualization, 'RGB')
```

Corylus avellana_anomalous GradCAM:

```
rgb_img = Image.open(path2).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
#cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
# cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=False)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
# plot Corylus avellana_anomalous GradCAMPlusPlus
print('Corylus avellana_anomalous GradCAMPlusPlus')
Image.fromarray(visualization, 'RGB')
```

Corylus avellana_anomalous GradCAMPlusPlus

```
rgb_img = Image.open(path2).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
#cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
#cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=True)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
100%|██████████| 32/32 [00:01<00:00, 24.89it/s]
```

```
# plot Corylus avellana_anomalous ScoreCAM
print('Corylus avellana_anomalous ScoreCAM:')
Image.fromarray(visualization, 'RGB')
```

Corylus avellana_anomalous ScoreCAM:

```
path3 = ('/content/MyPollen13K/train/3/20190404105005_OBJ_0_933_905.png')
print('Alnus__well developed:')
Image.open(path3).convert('RGB')
```

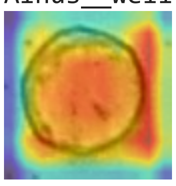
Alnus__well developed:

```
rgb_img = Image.open(path3).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!

# Construct the CAM object once, and then re-use it on many images:
cam1 = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
#cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
# cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=False)

# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...

# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None

# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam1 = cam1(input_tensor=input_tensor)
```

```
# In this example grayscale_cam1 has only one image in the batch:
grayscale_cam1 = grayscale_cam1[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam1, use_rgb=True)
```

```
# plot Alnus__well developed GradCAM
print('Alnus__well developed GradCAM:')
Image.fromarray(visualization, 'RGB')
```

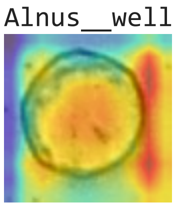
```
Alnus__well developed GradCAM:
```



```
rgb_img = Image.open(path3).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
#cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
# cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=False)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
# plot Alnus__well developed GradCAMPlusPlus
print('Alnus__well developed GradCAMPlusPlus')
Image.fromarray(visualization, 'RGB')
```

```
Alnus__well developed GradCAMPlusPlus
```



```
rgb_img = Image.open(path3).convert('RGB')
# Max min normalization
rgb_img = (rgb_img - np.min(rgb_img)) / (np.max(rgb_img) - np.min(rgb_img))
# Create an input tensor image for your model
input_tensor = torchvision.transforms.functional.to_tensor(rgb_img).unsqueeze(0).float()
# Note: input_tensor can be a batch tensor with several images!
# Construct the CAM object once, and then re-use it on many images:
#cam = GradCAM(model=model, target_layers=target_layers, use_cuda=True)
#cam = GradCAMPlusPlus(model=model, target_layers=target_layers, use_cuda=True)
cam = ScoreCAM(model=model, target_layers=target_layers, use_cuda=True)
# You can also use it within a with statement, to make sure it is freed,
# In case you need to re-create it inside an outer loop:
# with GradCAM(model=model, target_layers=target_layers, use_cuda=args.use_cuda) as cam:
#   ...
# We have to specify the target we want to generate
# the Class Activation Maps for.
# If targets is None, the highest scoring category
# will be used for every image in the batch.
# Here we use ClassifierOutputTarget, but you can define your own custom targets
# That are, for example, combinations of categories, or specific outputs in a non standard model.
# targets = [e.g ClassifierOutputTarget(281)]
# target_category = None
# You can also pass aug_smooth=True and eigen_smooth=True, to apply smoothing.
grayscale_cam = cam(input_tensor=input_tensor)
# In this example grayscale_cam has only one image in the batch:
grayscale_cam = grayscale_cam[0, :]
visualization = show_cam_on_image(rgb_img, grayscale_cam, use_rgb=True)
```

```
100%|██████████| 32/32 [00:01<00:00, 25.45it/s]
```

```
# plot Alnus__well developed ScoreCAM
print('Alnus__well developed ScoreCAM:')
Image.fromarray(visualization, 'RGB')
```

```
Alnus__well developed ScoreCAM:
```