Fantastic µPlastic Machine

•••

Technology Review

Will Ballengee, Nida Janulaitis, Samantha Phan, Liwen Xing

Background

Sci. Total Environ. 737 (2020) 140279

aarp.org

- Microplastic pollution is a global concern.
- Can computer vision and machine learning facilitate the study of microplastic pollution?
- **Data:** Microplastic and non-microplastic images that have been chemically analyzed
- **Goal:** Develop a ML model to measure size and categorize shape and color of microplastic; potentially distinguish between microplastic and non-microplastic

Considered Technologies

TensorFlow

- Open source machine learning library made by Google
- Very popular and well documented

K Keras

- Built on top of TensorFlow
- High-level
- Easy to use but less access to inner-workings of machine learning models

scikit learn

- High-level machine learning library
- ‘Out of the box’ algorithms
- Image processing tools

PyTorch

Appeal and Drawbacks of PyTorch

Pros

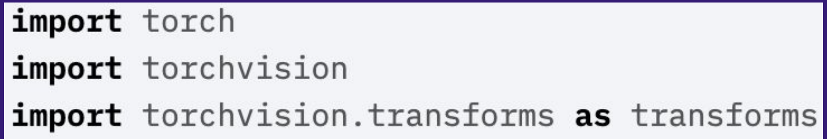
- Intuitive
- Easy to pick up
- Efficient training times
- Popular choice for computer vision applications
- Easy debugging with python debugger
- Parallelization
- Integrates well with other libraries (e.g. numpy)

Cons

- Newer/ less widely used
- Lacking documentation

Pytorch and how it will work for this project

PyTorch is an open source machine learning library based on the Torch library

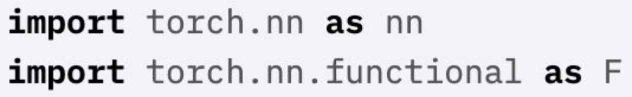


```
import torch
import torchvision
import torchvision.transforms as transforms
```

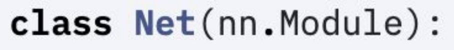
For vision specifically

Training an image classifier using Pytorch:

- Load and normalizing the training and test datasets using torchvision
- Define a Convolutional Neural Network



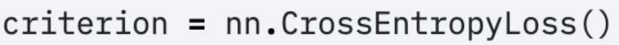
```
import torch.nn as nn
import torch.nn.functional as F
```



```
class Net(nn.Module):
```

Defined functions

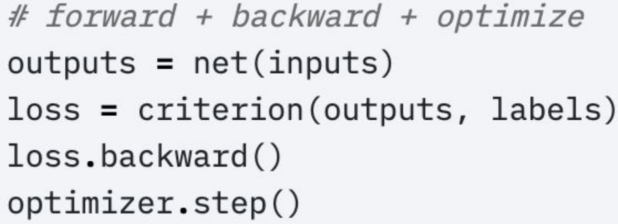
- Define a loss function



```
criterion = nn.CrossEntropyLoss()
```

- Train the network on the training data

Feed the inputs to the network using data iterator and optimize (Forward and Backward)



```
# forward + backward + optimize
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

- Save the trained network and test it on the test data