resnetcnn

December 3, 2025

[4]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split, Dataset
import os
```

[5]:
```
np.random.seed(0)
```

[6]:
```
if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
elif torch.cuda.is_available():
    device = torch.device("cpu")
    use_mps = False

print(device)
```

```
mps
```

[7]:
```
class PLKDataset(Dataset):
    def __init__(self, file_path, transform=None):
        with open(file_path, 'rb') as f:
            data = pickle.load(f)
        self.images = data['images']
        self.labels = data['labels'].reshape(-1)
        self.transform = transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image = self.images[idx]
        label = self.labels[idx]
```

```
        if self.transform:
            image = self.transform(image)
        return image, label
```

[8]:
```
dataset = PLKDataset('ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
↪pkl')
```

[9]:
```
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

[10]:
```
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

[11]:
```
from sklearn.model_selection import train_test_split

labels = dataset.labels
idx = np.arange(len(dataset))
train_idx, valid_idx = train_test_split(idx, test_size=0.2, stratify=labels,␣
↪random_state=42)
```

[12]:
```
class TransformSubset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, idx):
        image, label = self.subset[idx]
        if self.transform:
            image = self.transform(image)
        return image, label

    def __len__(self):
        return len(self.subset)
```

[13]:
```
from torch.utils.data import Subset

train_dataset = Subset(dataset, train_idx)
```

```
train_data = TransformSubset(train_dataset, train_transform)

val_dataset = Subset(dataset, valid_idx)
val_data = TransformSubset(val_dataset, val_transform)

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
val_loader = DataLoader(val_data, batch_size=32, shuffle=False)
```

[14]:
```
model = models.resnet50(weights="IMAGENET1K_V2")
```

```
Downloading: "https://download.pytorch.org/models/resnet50-11ad3fa6.pth" to
/Users/yamira.poldosilva/.cache/torch/hub/checkpoints/resnet50-11ad3fa6.pth

100%|      | 97.8M/97.8M [00:01<00:00, 62.3MB/s]
```

[15]:
```
num_classes = 5

model.fc = nn.Linear(model.fc.in_features, num_classes)
```

[16]:
```
for param in model.parameters():
    param.requires_grad = False

for param in model.fc.parameters():
    param.requires_grad = True
```

[17]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
```

[18]:
```
for epoch in range(5):
    model.train()

    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[18], line 4
      1 for epoch in range(5):
      2     model.train()
----> 4     for images, labels in train_loader:
      5         optimizer.zero_grad()
      6         outputs = model(images)
```

```
File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/dataloader.py:732, in _BaseDataLoaderIter.__next__(self)
    729 if self._sampler_iter is None:
    730     # TODO(https://github.com/pytorch/pytorch/issues/76750)
    731     self._reset()  # type: ignore[call-arg]
--> 732 data = self._next_data()
    733 self._num_yielded += 1
    734 if (
    735     self._dataset_kind == _DatasetKind.Iterable
    736     and self._IterableDataset_len_called is not None
    737     and self._num_yielded > self._IterableDataset_len_called
    738 ):

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/dataloader.py:788, in _SingleProcessDataLoaderIter.
 ↪_next_data(self)
    786 def _next_data(self):
    787     index = self._next_index()  # may raise StopIteration
--> 788     data = self._dataset_fetcher.fetch(index)  # may raise StopIteratio
    789     if self._pin_memory:
    790         data = _utils.pin_memory.pin_memory(data, self.
 ↪_pin_memory_device)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/utils/data/_utils/fetch.py:52, in _MapDatasetFetcher.fetch(self,␣
 ↪possibly_batched_index)
     50         data = self.dataset.__getitems__(possibly_batched_index)
     51     else:
---> 52         data = [self.dataset[idx] for idx in possibly_batched_index]
     53 else:
     54     data = self.dataset[possibly_batched_index]

Cell In[12], line 9, in TransformSubset.__getitem__(self, idx)
      7 image, label = self.subset[idx]
      8 if self.transform:
----> 9     image = self.transform(image)
     10 return image, label

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torchvision/transforms/transforms.py:95, in Compose.__call__(self, img)
     93 def __call__(self, img):
     94     for t in self.transforms:
---> 95         img = t(img)
     96     return img
```

```
File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/nn/modules/module.py:1775, in Module._wrapped_call_impl(self, *args,␣
 ↪**kwargs)
   1773     return self._compiled_call_impl(*args, **kwargs)  # type:␣
 ↪ignore[misc]
   1774 else:
-> 1775     return self._call_impl(*args, **kwargs)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torch/nn/modules/module.py:1786, in Module._call_impl(self, *args, **kwargs)
   1781 # If we don't have any hooks, we want to skip the rest of the logic in
   1782 # this function, and just call forward.
   1783 if not (self._backward_hooks or self._backward_pre_hooks or self.
 ↪_forward_hooks or self._forward_pre_hooks
   1784         or _global_backward_pre_hooks or _global_backward_hooks
   1785         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1786     return forward_call(*args, **kwargs)
   1788 result = None
   1789 called_always_called_hooks = set()

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torchvision/transforms/transforms.py:354, in Resize.forward(self, img)
    346 def forward(self, img):
    347     """
    348     Args:
    349         img (PIL Image or Tensor): Image to be scaled.
   (…)    352         PIL Image or Tensor: Rescaled image.
    353     """
--> 354     return␣
 ↪F.resize(img, self.size, self.interpolation, self.max_size, self.antialias)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torchvision/transforms/functional.py:465, in resize(img, size, interpolation,␣
 ↪max_size, antialias)
    459     if max_size is not None and len(size) != 1:
    460         raise ValueError(
    461             "max_size should only be passed if size specifies the lengt␣
 ↪of the smaller edge, "
    462             "i.e. size should be an int or a sequence of length 1 in␣
 ↪torchscript mode."
    463         )
--> 465 _, image_height, image_width = get_dimensions(img)
    466 if isinstance(size, int):
    467     size = [size]

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torchvision/transforms/functional.py:80, in get_dimensions(img)
     77 if isinstance(img, torch.Tensor):
```

```
     78     return F_t.get_dimensions(img)
---> 80 return F_pil.get_dimensions(img)

File ~/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.13/site-packages/
 ↪torchvision/transforms/_functional_pil.py:34, in get_dimensions(img)
     32     width, height = img.size
     33     return [channels, height, width]
---> 34 raise TypeError(f"Unexpected type {type(img)}")

TypeError: Unexpected type <class 'numpy.ndarray'>
```

[ ]:
```
model.eval()
correct = 0
total = 0

with torch.no_grad():
    for images, labels in val_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print("Accuracy on validation set: {:.2f}%".format(100 * correct / total))
```