ants-bees

March 16, 2025

0.1 Distinguishing ants from bees

This notebook builds a classifier that distinguishes between images of ants and bees. The classifier has three parts to it: - The images are of varying sizes. So first, they are all normalized to a fixed size. - Then they are run through a pre-trained computer vision neural net, ResNet50, that produces a 2048-dimensional representation - Finally, a logistic regression classifier is built on top of this representation.

0.1.1 Various includes

[20]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
# Torch stuff
import torch
import torch.nn as nn
# Torchvision stuff
from torchvision import datasets, models, transforms
# sklearn stuff
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,accuracy_score
```

0.1.2 Loading Dataset

For both the train and test data, the images need to be normalized to the particular size, 224x224x3, that is required by the ResNet50 network that we will apply to them. This is achieved by a series of transforms.

- The (normalized) training set is in image_datasets['train']
- The (normalized) test set is in image_datasets['val']

[14]:
```
data_transforms = {
    'train': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        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])
    ]),
}

data_dir = './hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),␣
 ↪data_transforms[x])
                  for x in ['train', 'val']}
```

Look at the classes and data set sizes

[15]:
```
class_names = image_datasets['train'].classes
class_names
```

[15]:
```
['ants', 'bees']
```

[16]:
```
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
dataset_sizes
```

[16]:
```
{'train': 244, 'val': 153}
```

1 Problem 6a

[21]:
```
image_path = image_datasets['train'].samples[item][0]
original_img = mpimg.imread(image_path)
print("Original Image:")
plt.imshow(original_img)
plt.axis('off')
plt.show()
```

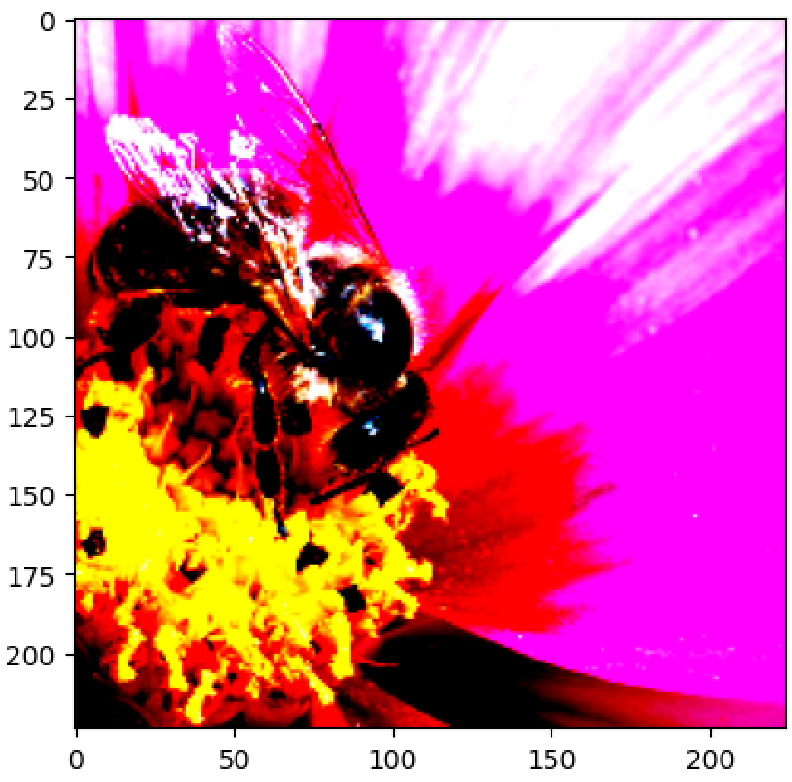
```
Original Image:
```

Print a sample (transformed) image

[17]:
```
item = 200
[itemx,itemy] = image_datasets['train'].__getitem__(item)
print("Label: {}\n".format(class_names[itemy]))
plt.imshow(itemx.permute(1, 2, 0))
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-2.0665298..2.64].
```

```
Label: bees
```



1.0.1 Load pre-trained ResNet50

Torch has a bunch of pre-trained nets for computer vision. Let's try out one of them: ResNet50.

[6]:
```
resnet50 = models.resnet50(pretrained = True)
modules = list(resnet50.children())[:-1]
resnet50 = nn.Sequential(*modules)
for p in resnet50.parameters():
    p.requires_grad = False
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to
```

```
/Users/dannyxia/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100.0%
```

1.0.2 Extract ResNet features from dataset

We'll use ResNet to produce a 2048-dimensional representation for each image.

The resulting training set will be in the Numpy arrays (X_train, y_train) and the test set will be in the Numpy arrays (X_test, y_test).

[7]:
```
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x])
              for x in ['train', 'val']}
for batch,data in enumerate(dataloaders['train']):
    if batch==0:
        X_train = torch.squeeze(resnet50(data[0])).numpy()
        y_train = data[1].numpy()
    else:
        X_train = np.vstack((X_train,torch.squeeze(resnet50(data[0])).numpy()))
        y_train = np.hstack((y_train,data[1].numpy()))


for batch,data in enumerate(dataloaders['val']):
    if batch==0:
        X_test = torch.squeeze(resnet50(data[0])).numpy()
        y_test = data[1].numpy()
    else:
        X_test = np.vstack((X_test,torch.squeeze(resnet50(data[0])).numpy()))
        y_test = np.hstack((y_test,data[1].numpy()))
```

[8]:
```
np.shape(X_train), np.shape(y_train), np.shape(X_test), np.shape(y_test)
```

[8]:
```
((244, 2048), (244,), (153, 2048), (153,))
```

1.0.3 Train logistic regression classifier on the ResNet features

And then we'll evaluate its performance on the test set.

[9]:
```
clf = LogisticRegression(solver='liblinear',random_state=0,max_iter=1000)
clf.fit(X_train, y_train)
```

[9]:
```
LogisticRegression(max_iter=1000, random_state=0, solver='liblinear')
```

[10]:
```
y_pred = clf.predict(X_test)
print("Accuracy: {}\n".format(accuracy_score(y_test,y_pred)))
print("Confusion matrix: \n {}".format(confusion_matrix(y_test,y_pred)))
```

```
Accuracy: 0.803921568627451

Confusion matrix:
 [[60 10]
 [20 63]]
```

[12]:
```
from sklearn.neighbors import KNeighborsClassifier

for k in [1, 3, 5]:
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='kd_tree')
    knn.fit(X_train, y_train)
    acc = knn.score(X_test, y_test)
    print(f"k-NN (k={k}) Accuracy: {acc:.4f}")
```

```
k-NN (k=1) Accuracy: 0.6928
k-NN (k=3) Accuracy: 0.7124
k-NN (k=5) Accuracy: 0.6928
```

[ ]:
```

```