Assignment_4

November 25, 2024

[17]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import matplotlib.pyplot as plt
```

Part 1

Data is loaded and two classes (frogs and planes) are isolated and reduced to 2000 images each.

[3]:
```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True)
trainloader = torch.utils.data.DataLoader(trainset,
                                          shuffle=True, num_workers=1)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True)
testloader = torch.utils.data.DataLoader(testset,
                                         shuffle=False, num_workers=1)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[4]:
```
cifar = trainset.data
targets = trainset.targets
labels = trainset.classes
```

[5]:
```
frogsAll = []
for i, img in enumerate(cifar):
    if targets[i] == 6:
        frogsAll.append(img)

frogs = frogsAll[:2000]
```

```
planesAll = []
for i, img in enumerate(cifar):
    if targets[i] == 0:
        planesAll.append(img)

planes = planesAll[:2000]
```

Part 2

The 10x10 cropping requirement was overlooked until after means and standard deviations had been calculated, but these values can still be used due to the independce of pixel values in GBM. Cropping will happen later in the model.

Part 3

Means and standard deviations are calculated stored in .npy array files (file storing lines have been removed from the code to prevent errors). After these calculations, the model has been trained.

[ ]:
```
#@title Mean/SD Calc
### Calculating means and standard deviations -- all saved to .npy files␣
 ↪(removed from code)
### Initially computed with 32x32 dimension, then GBM uses only middle 10x10
### Can use 32x32 values because each pixel is independent of others

# Frog means

frog_means = np.ndarray(shape=(32,32,3), dtype=float)


for i, pic in enumerate(frogs):
  for j, row in enumerate(pic):
    for k, pixel in enumerate(row):
      for l, colorVal in enumerate(pixel):
        frog_means[j,k,l] += colorVal

frog_means /= 2000.0

# Plane means

planes_means = np.zeros(shape=(32,32,3), dtype=float)

for i, pic in enumerate(planes):
  for j, row in enumerate(pic):
    for k, pixel in enumerate(row):
      for l, colorVal in enumerate(pixel):
        planes_means[j,k,l] += colorVal

planes_means /= 2000.0
```

```
# Frog standard deviations

frog_means = np.load("frog_means.npy")
frog_sds = np.ndarray(shape=(32,32,3), dtype=float)

for j, row in enumerate(frog_means):
  for k, pixel in enumerate(row):
    for l, colorVal in enumerate(pixel):
      frog_sds[j,k,l] = 0.0
      for i, pic in enumerate(frogs):
        frog_sds[j,k,l] += ((pic[j,k,l] - frog_means[j,k,l])**2)/2000.0
      frog_sds[j,k,l] = np.sqrt(frog_sds[j,k,l])

# Plane standard deviations

planes_means = np.load("planes_means.npy")
planes_sds = np.zeros(shape=(32,32,3), dtype=float)

for j, row in enumerate(planes_means):
  for k, pixel in enumerate(row):
    for l, colorVal in enumerate(pixel):
      planes_sds[j,k,l] = 0.0
      for i, pic in enumerate(planes):
        planes_sds[j,k,l] += ((pic[j,k,l] - planes_means[j,k,l])**2)/2000.0
      planes_sds[j,k,l] = np.sqrt(planes_sds[j,k,l])
```

The means and standard deviations are loaded and cropped to 10x10. The frogs and planes from the test set are isolated and cropped to 10x10. An accuracy test is performed for both classes.

Frogs: (171/200) 85.5%

Planes: (160/200) 80.0%

[36]:
```
# load computed means and sds and isolate center 10x10
frog_means = np.load("frog_means.npy")[11:21, 11:21, :]
frog_sds = np.load("frog_sds.npy")[11:21, 11:21, :]
planes_means = np.load("planes_means.npy")[11:21, 11:21, :]
planes_sds = np.load("planes_sds.npy")[11:21, 11:21, :]

# isolate/convert test set
test_frogs = []
test_planes = []
for i, target in enumerate(testset.targets):
  if target == 6:
    test_frogs.append(testset.data[i])
  elif target == 0:
    test_planes.append(testset.data[i])
```

```
test_frogs = np.array(test_frogs)[:200, 11:21, 11:21, :]
test_planes = np.array(test_planes)[:200, 11:21, 11:21, :]

test_imgs = np.concatenate((test_frogs, test_planes))

# accuracy test - frogs and planes
corr_frogs = 0
corr_planes = 0
zeros = 0
for index, input in enumerate(test_imgs):
  frog_pdfs = np.zeros(shape=(10,10,3), dtype=float)
  plane_pdfs = np.zeros(shape=(10,10,3), dtype=float)
  for i, row in enumerate(input):
    for j, pixel in enumerate(row):
      for k, colorVal in enumerate(pixel):
        frog_pdfs[i,j,k] = norm.pdf(colorVal, loc=frog_means[i,j,k],␣
 ↪scale=frog_sds[i,j,k])
        plane_pdfs[i,j,k] = norm.pdf(colorVal, loc=planes_means[i,j,k],␣
 ↪scale=planes_sds[i,j,k])
  frog_prob = np.sum(np.log(frog_pdfs)) + np.log(.5) # logistic transformation
  plane_prob = np.sum(np.log(plane_pdfs)) + np.log(.5)
  if frog_prob > plane_prob and index < 200:
    corr_frogs += 1
  elif plane_prob > frog_prob and index >= 200:
    corr_planes += 1

print(str(corr_frogs/(len(test_imgs)/2)*100) + "% correct (" + str(corr_frogs)␣
 ↪+ ")")
print(str(corr_planes/(len(test_imgs)/2)*100) + "% correct (" +␣
 ↪str(corr_planes) + ")")
```

```
85.5% correct (171)
80.0% correct (160)
```

Part 4

Results are plotted in a stacked bar chart, showing accuracy for both classes across the 400 tests.

[35]:
```
class_names = (
    "Frog (85.5%)",
    "Plane (80.0%)",
)
accuracy = {
    "Correct": np.array([corr_frogs, corr_planes]),
    "Incorrect": np.array([200-corr_frogs, 200-corr_planes]),
}
width = 0.7
```

```
fig, ax = plt.subplots()
bottom = np.zeros(2)

for boolean, value in accuracy.items():
    p = ax.bar(class_names, value, width, label=boolean, bottom=bottom)
    bottom += value

ax.set_title("Accuracy of Predictions by Ground Truth")
ax.legend(loc="upper right")

plt.show()
```