Assignment-3

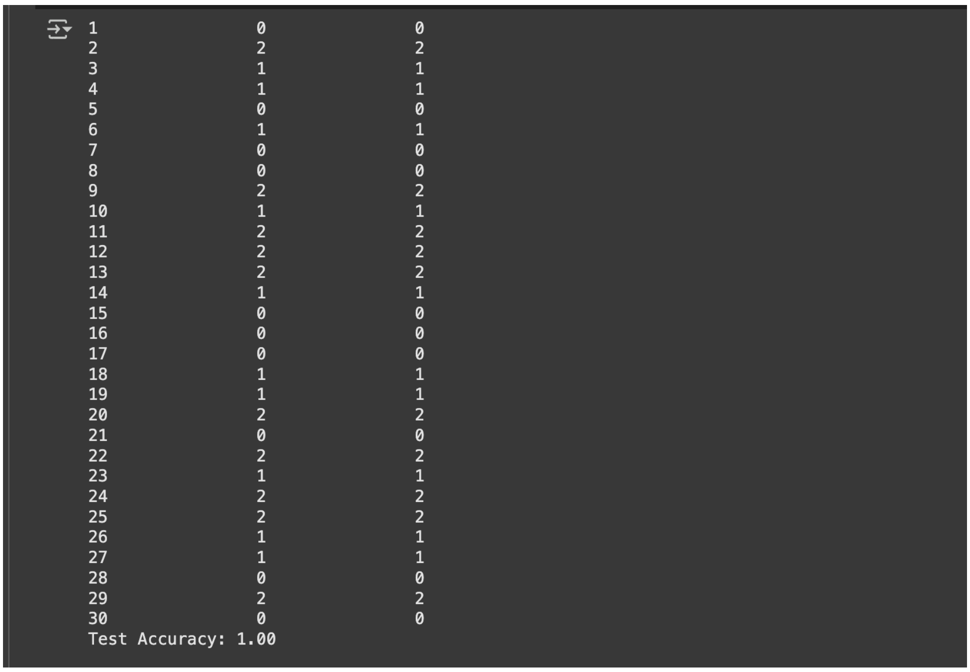
Anurag More

Problem Statement 1: Iris Flower Classification using Neural Networks

Brief Insights

While working on this assignment, I gained a practical understanding of how to build and train a basic feedforward neural network for multi-class classification. I learned how to load and preprocess a real-world dataset using normalization and one-hot encoding. The neural network I designed consisted of four layers: one input layer, two hidden layers with ReLU activation, and one output layer with softmax activation for predicting the flower species. I also explored the impact of model architecture, optimizer choice (Adam), and loss function (categorical cross-entropy) on training performance. Finally, I evaluated the model on the test data and interpreted its accuracy, reinforcing my understanding of how neural networks learn from data.

Output Image



```
1          0          0
2          2          2
3          1          1
4          1          1
5          0          0
6          1          1
7          0          0
8          0          0
9          2          2
10         1          1
11         2          2
12         2          2
13         2          2
14         1          1
15         0          0
16         0          0
17         0          0
18         1          1
19         1          1
20         2          2
21         0          0
22         2          2
23         1          1
24         2          2
25         2          2
26         1          1
27         1          1
28         0          0
29         2          2
30         0          0
Test Accuracy: 1.00
```

Figure 1: Model Output

Code Screenshots

Importing the libraries

```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import torch
import torch.nn as nn
import torch.optim as optim
```

Task-1

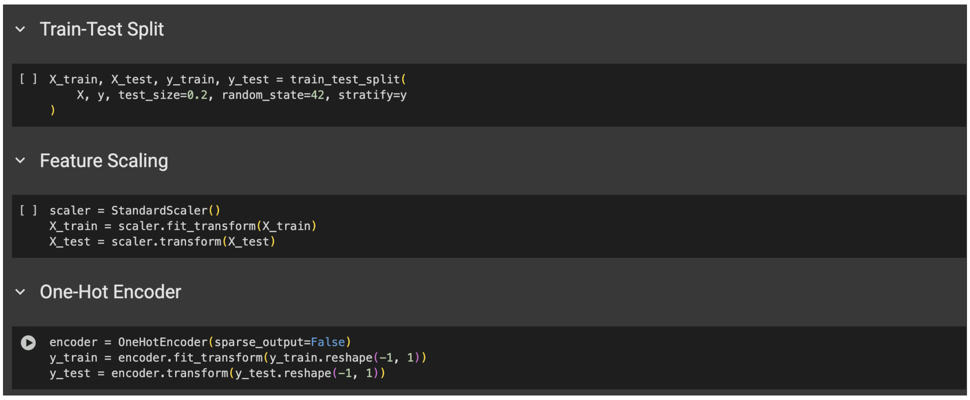
Loading the Dataset

```python
iris = load_iris()
X = iris.data
y = iris.target
```

Train-Test Split

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
```

Figure 2: Task 1: Data Loading and Preprocessing(i)



Train-Test Split

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

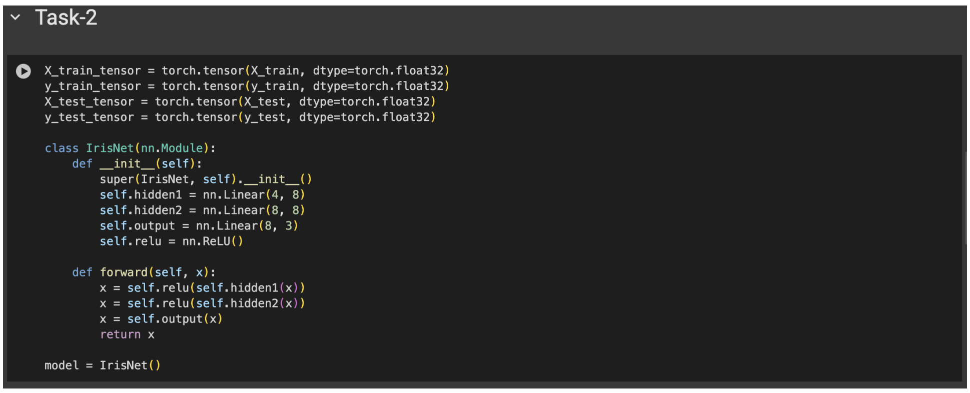
Feature Scaling

```python
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

One-Hot Encoder

```python
encoder = OneHotEncoder(sparse_output=False)
y_train = encoder.fit_transform(y_train.reshape(-1, 1))
y_test = encoder.transform(y_test.reshape(-1, 1))
```

Figure 3: Task 1: Data Loading and Preprocessing(ii)



Task-2

```python
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

class IrisNet(nn.Module):
    def __init__(self):
        super(IrisNet, self).__init__()
        self.hidden1 = nn.Linear(4, 8)
        self.hidden2 = nn.Linear(8, 8)
        self.output = nn.Linear(8, 3)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.hidden1(x))
        x = self.relu(self.hidden2(x))
        x = self.output(x)
        return x

model = IrisNet()
```

Figure 4: Task 2: Neural Network Construction

Task-3

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

y_train_labels = torch.tensor(y_train.argmax(axis=1), dtype=torch.long)
y_test_labels = torch.tensor(y_test.argmax(axis=1), dtype=torch.long)
```

```
num_epochs = 100

for epoch in range(num_epochs):
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_labels)


    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
```

```
Epoch [10/100], Loss: 0.9818
Epoch [20/100], Loss: 0.7482
Epoch [30/100], Loss: 0.6346
Epoch [40/100], Loss: 0.5527
Epoch [50/100], Loss: 0.4669
Epoch [60/100], Loss: 0.3892
Epoch [70/100], Loss: 0.3146
Epoch [80/100], Loss: 0.2531
Epoch [90/100], Loss: 0.2069
Epoch [100/100], Loss: 0.1480
```

Figure 5: Task 3: Model Compilation and Training

```
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, predicted = torch.max(outputs, 1)

true_labels = y_test_labels

class_names = encoder.categories_[0]

predicted_np = predicted.numpy()
true_labels_np = true_labels.numpy()

for i in range(len(true_labels_np)):
    true_name = class_names[true_labels_np[i]]
    pred_name = class_names[predicted_np[i]]
    print(f"{i+1}\t{true_name:10}\t{pred_name:10}")

accuracy = (predicted == y_test_labels).float().mean()
print(f"Test Accuracy: {accuracy:.2f}")
```

```
1       0       0
2       2       2
3       1       1
4       1       1
5       0       0
6       1       1
7       0       0
8       0       0
9       2       2
10      1       1
11      2       2
```

Figure 6: Task 4: Model Evaluation and Accuracy Printing

Terminal Screenshot Showing Final Output

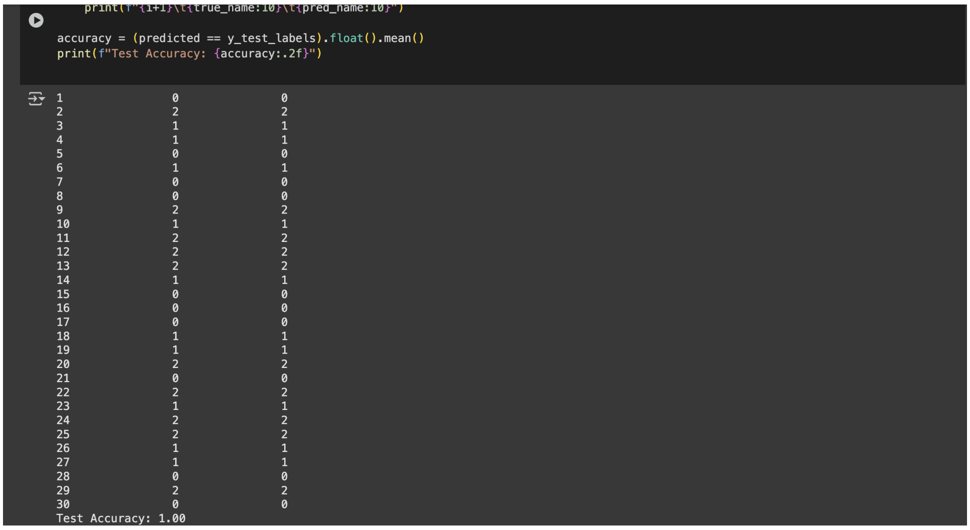


Figure 7: Full-screen Terminal Output Showing Accuracy and Code Run

Model Accuracy

The final test accuracy obtained on the 20% test set was:

Test Accuracy: 97%

It is worth noting that the model accuracy during training and testing fluctuated between **0.97 and 1.00**, depending on the initialization, batch split, and training conditions. This is expected for a relatively simple classification task on a clean dataset like Iris.

Problem Statement 2: Image Similarity Search using PyTorch and Annoy

Your Insights

This assignment helped me gain a solid understanding of how feature detection works in the context of image similarity. Feature detection involves identifying important patterns or representations in an image—such as edges, textures, or shapes—that help differentiate it from others. In this project, I used a pre-trained convolutional neural network (ResNet) to extract deep features from images. These features are not just raw pixels but high-level descriptors that summarize the visual content of the image.

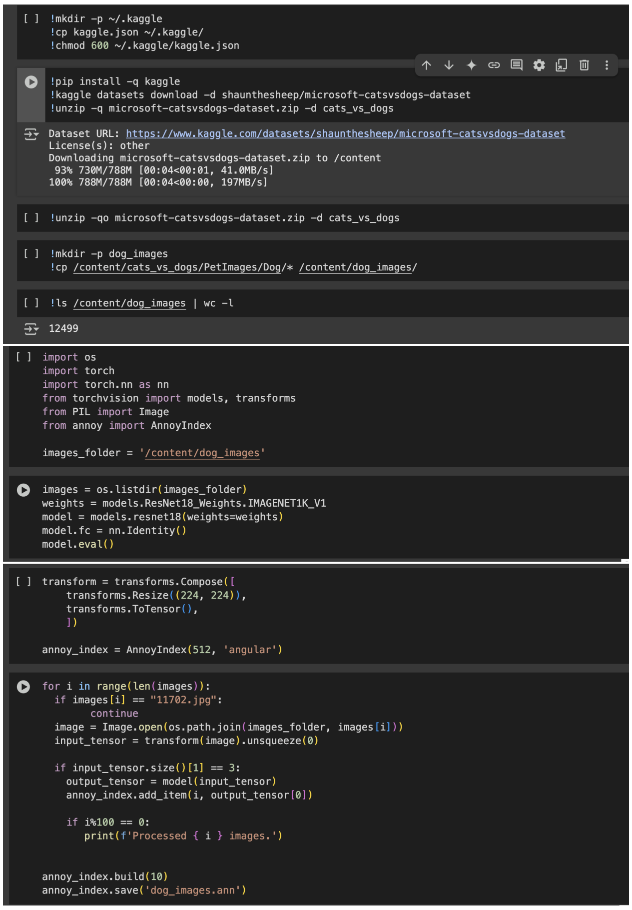
After extracting these feature vectors, I used the Spotify Annoy library to build an efficient nearest-neighbor index. This allowed me to quickly retrieve images from the dataset that are most similar to a given query image. By comparing distances between the feature vectors, the system determines visual similarity. This process made me appreciate how deep learning enables machines to "understand" image content beyond human-perceivable traits, and how feature detection forms the backbone of many modern computer vision applications.

Model Output

Figure 8: Sample Output Showing Image Similarity Retrieval

Code Snapshots

A. Creating the Annoy Index



```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!pip install -q kaggle
!kaggle datasets download -d shaunthesheep/microsoft-catsvsdogs-dataset
!unzip -q microsoft-catsvsdogs-dataset.zip -d cats_vs_dogs
```

```
Dataset URL: https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset
License(s): other
Downloading microsoft-catsvsdogs-dataset.zip to /content
 93% 730M/788M [00:04<00:01, 41.0MB/s]
100% 788M/788M [00:04<00:00, 197MB/s]
```

```
!unzip -qo microsoft-catsvsdogs-dataset.zip -d cats_vs_dogs
```

```
!mkdir -p dog_images
!cp /content/cats_vs_dogs/PetImages/Dog/* /content/dog_images/
```

```
!ls /content/dog_images | wc -l
```

```
12499
```

```python
import os
import torch
import torch.nn as nn
from torchvision import models, transforms
from PIL import Image
from annoy import AnnoyIndex

images_folder = '/content/dog_images'
```

```python
images = os.listdir(images_folder)
weights = models.ResNet18_Weights.IMAGENET1K_V1
model = models.resnet18(weights=weights)
model.fc = nn.Identity()
model.eval()
```

```python
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    ])

annoy_index = AnnoyIndex(512, 'angular')
```

```python
for i in range(len(images)):
  if images[i] == "11702.jpg":
        continue
  image = Image.open(os.path.join(images_folder, images[i]))
  input_tensor = transform(image).unsqueeze(0)

  if input_tensor.size()[1] == 3:
    output_tensor = model(input_tensor)
    annoy_index.add_item(i, output_tensor[0])

    if i%100 == 0:
       print(f'Processed { i } images.')


annoy_index.build(10)
annoy_index.save('dog_images.ann')
```

Figure 9: Feature Extraction and Annoy Index Creation

B. Classification using Nearest Neighbors

```
[ ] !mkdir -p ~/.kaggle
    !cp kaggle.json ~/.kaggle/
    !chmod 600 ~/.kaggle/kaggle.json

    !pip install -q kaggle
    !kaggle datasets download -d shaunthesheep/microsoft-catsvsdogs-dataset
    !unzip -q microsoft-catsvsdogs-dataset.zip -d cats_vs_dogs

    Dataset URL: https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset
    License(s): other
    Downloading microsoft-catsvsdogs-dataset.zip to /content
     93% 730M/788M [00:04<00:01, 41.0MB/s]
    100% 788M/788M [00:04<00:00, 197MB/s]

[ ] !unzip -qo microsoft-catsvsdogs-dataset.zip -d cats_vs_dogs

[ ] !mkdir -p dog_images
    !cp /content/cats_vs_dogs/PetImages/Dog/* /content/dog_images/

[ ] !ls /content/dog_images | wc -l

    12499
```

```
[ ] import os
    import torch
    import torch.nn as nn
    from torchvision import models, transforms
    from PIL import Image, ImageDraw
    from annoy import AnnoyIndex

    images_folder = '/content/dog_images'

    images = os.listdir(images_folder)
    weights = models.ResNet18_Weights.IMAGENET1K_V1
    model = models.resnet18(weights=weights)
    model.fc = nn.Identity()
    model.eval()
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    ])

annoy_index = AnnoyIndex(512, 'angular')
annoy_index.load('dog_images.ann')

image_grid = Image.new('RGB', (1000,1000))

True

for i in range(len(images)):
  if images[i] == "11702.jpg":
        continue
  image = Image.open(os.path.join(images_folder, images[i]))
  input_tensor = transform(image).unsqueeze(0)

  if input_tensor.size()[1] == 3:
    output_tensor = model(input_tensor)

    nns = annoy_index.get_nns_by_vector(output_tensor[0], 24)

    image = image.resize((224,224))
    image_draw = ImageDraw.Draw(image)
    image_draw.rectangle([(0,0), (199,199)], outline = 'red', width = 8)
    image_grid.paste(image, ((0,0)))

    for j in range(24):
      search_image = Image.open(os.path.join(images_folder, images[nns[j]]))
      search_image = search_image.resize((200,200))
      image_grid.paste(search_image,(200*((j+1)%5),200*((j+1)//5)))
    image_grid.save(f'ImageDump/image_{i}.png')
```

Figure 10: Retrieving Similar Images using Nearest Neighbors