

```
In [37]: print ('بسم الله الرحمن الرحيم')
print ('الاختبار الاول لمادة')
print ('Practical_Image_Processing_and_Natural_Language_Processing_Second Trimes')
print ('تقديم الطالب / حسن محمد احمد الامير خليفة')
print ('الرقم الجامعي', '253002941')
print ('اعداد الدكتور / هاجر صالح')
print ('https://www.kaggle.com/datasets/sanikamal/rock-paper-
```

بسم الله الرحمن الرحيم
الاختبار الاول لمادة
Practical_Image_Processing_and_Natural_Language_Processing_Second Trimester
تقديم الطالب / حسن محمد احمد الامير خليفة
الرقم الجامعي 253002941
اعداد الدكتور / هاجر صالح
https://www.kaggle.com/datasets/sanikamal/rock-paper-scissors-dataset

```
In [2]: print ('Import Libraries')
import os
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split
import torch.nn as nn
import torch.optim as optim
```

Import Libraries

```
In [3]: print ('اختبار الجهاز')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(device)
```

اختبار الجهاز
cpu

```
In [20]: print ('تقليل حجم الصورة')
train_transform = transforms.Compose([

    transforms.Resize((128,128)),

    transforms.RandomHorizontalFlip(),

    transforms.RandomRotation(15),

    transforms.ToTensor()

])

test_transform = transforms.Compose([

    transforms.Resize((128,128)),

    transforms.ToTensor()

])
```

تقليل حجم الصورة

```
In [21]: print ( 'خطوة تحميل البيانات ' )
train_path = r"E:\حسن\Master AI\imagedataset\seg_train\seg_train"

test_path = r"E:\حسن\Master AI\imagedataset\seg_test\seg_test"

train_dataset = datasets.ImageFolder(
    root=train_path,
    transform=train_transform
)

test_dataset = datasets.ImageFolder(
    root=test_path,
    transform=test_transform
)
```

خطوة تحميل البيانات

```
In [22]: print ( 'خطوة معرفة الفئات ' )
print("Classes:")

print(train_dataset.classes)

print()

print("Number of Classes:")

print(len(train_dataset.classes))
```

خطوة معرفة الفئات

Classes:

['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']

Number of Classes:

6

```
In [23]: print ( 'إنشاء DataLoader' )
train_loader = DataLoader(
    train_dataset,
    batch_size=32,
    shuffle=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=32,
    shuffle=False
)
```

إنشاء DataLoader

```
In [15]: print("Training Images :", len(train_dataset))

print("Testing Images :", len(test_dataset))
```

Training Images : 14034

Testing Images : 3000

```
In [24]: images, labels = next(iter(train_loader))

print(images.shape)
print(labels.shape)
```

```
torch.Size([32, 3, 128, 128])
torch.Size([32])
```

```
In [25]: print ('لعرض اول ستة صور')
import matplotlib.pyplot as plt

classes = train_dataset.classes

plt.figure(figsize=(12,6))

for i in range(6):

    plt.subplot(2,3,i+1)

    img = images[i].permute(1,2,0)

    plt.imshow(img)

    plt.title(classes[labels[i]])

    plt.axis("off")

plt.show()
```

لعرض اول ستة صور



```
In [26]: print ('بناء السبي ان ان')
class CNN(nn.Module):

    def __init__(self):

        super(CNN,self).__init__()

        self.conv = nn.Sequential(

            nn.Conv2d(3,32,3,padding=1),

            nn.ReLU(),

            nn.MaxPool2d(2),
```

```

        nn.Conv2d(32,64,3,padding=1),

        nn.ReLU(),

        nn.MaxPool2d(2),

        nn.Conv2d(64,128,3,padding=1),

        nn.ReLU(),

        nn.MaxPool2d(2)

    )

    self.fc = nn.Sequential(

        nn.Flatten(),

        nn.Linear(128*16*16,256),

        nn.ReLU(),

        nn.Linear(256,6)

    )

    def forward(self,x):

        x=self.conv(x)

        x=self.fc(x)

        return x

```

بناء السي ان ان

```

In [27]: print ( 'انشاء بناء السي ان ان ' )
import torch.nn as nn

class CNN(nn.Module):

    def __init__(self):

        super(CNN, self).__init__()

        self.conv_layers = nn.Sequential(

            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)

```

```

    )

    self.fc_layers = nn.Sequential(

        nn.Flatten(),

        nn.Linear(128 * 16 * 16, 256),

        nn.ReLU(),

        nn.Dropout(0.5),

        nn.Linear(256, 6)

    )

    def forward(self, x):

        x = self.conv_layers(x)

        x = self.fc_layers(x)

        return x

```

انشاء بناء السي ان ان

```

In [12]: print ('انشاء النموذج')
         model = CNN().to(device)

         print(model)

```

انشاء النموذج

```

CNN(
  (conv_layers): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=32768, out_features=256, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=256, out_features=6, bias=True)
  )
)

```

```

In [28]: print ('اختيار Loss و Optimizer')
         criterion = nn.CrossEntropyLoss()

         optimizer = torch.optim.Adam(
             model.parameters(),

```

```
lr=0.001
)
```

Optimizer و Loss اختيار

```
In [29]: print ('مرحلة تدريب النموذج')
num_epochs = 5

train_losses = []

for epoch in range(num_epochs):

    model.train()

    running_loss = 0

    for images, labels in train_loader:

        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running_loss += loss.item()

    epoch_loss = running_loss / len(train_loader)

    train_losses.append(epoch_loss)

    print(f"Epoch [{epoch+1}/{num_epochs}] Loss: {epoch_loss:.4f}")
```

مرحلة تدريب النموذج

```
Epoch [1/5] Loss: 1.1583
Epoch [2/5] Loss: 0.9064
Epoch [3/5] Loss: 0.7967
Epoch [4/5] Loss: 0.6948
Epoch [5/5] Loss: 0.6426
```

```
In [30]: print ('مرحلة تقييم النموذج')
model.eval()

correct = 0

total = 0

with torch.no_grad():

    for images, labels in test_loader:

        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
```

```

_, predicted = torch.max(outputs, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total

print(f"Test Accuracy: {accuracy:.2f}%")

```

مرحلة تقييم النموذج

Test Accuracy: 80.00%

```

In [31]: print ('رسم منحنى التدريب')
plt.figure(figsize=(6,4))

plt.plot(train_losses, marker="o")

plt.title("Training Loss")

plt.xlabel("Epoch")

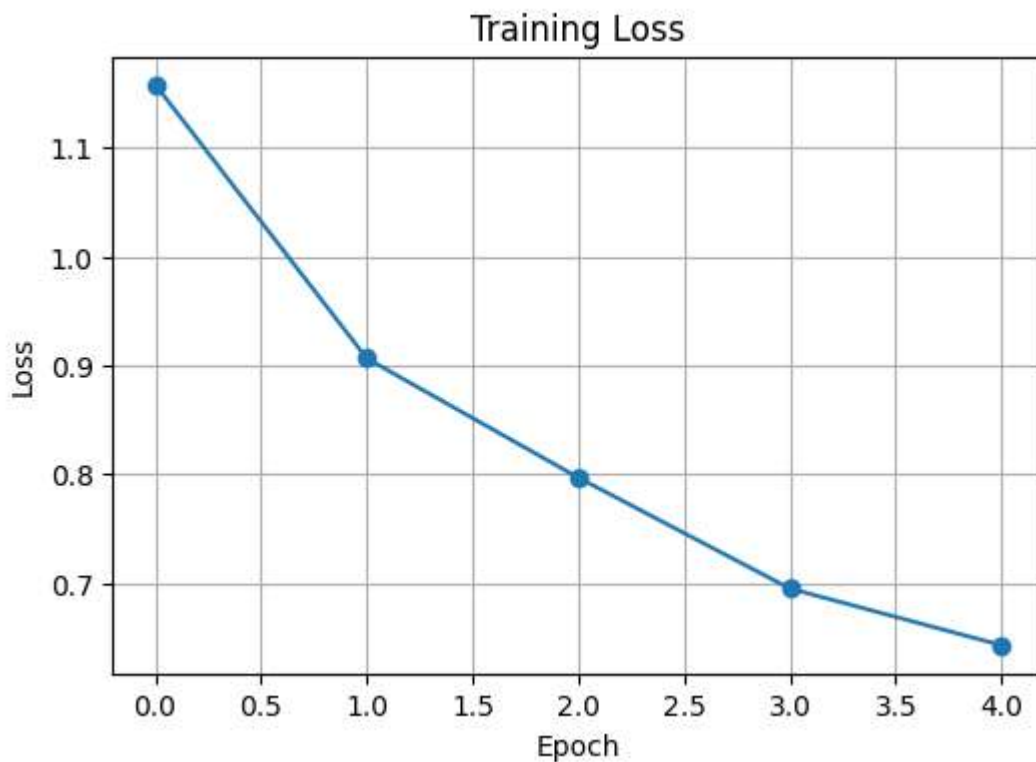
plt.ylabel("Loss")

plt.grid(True)

plt.show()

```

رسم منحنى التدريب



```

In [32]: print ('Confusion Matrix')
from sklearn.metrics import confusion_matrix
import seaborn as sns

all_labels = []
all_predictions = []

```

```

model.eval()

with torch.no_grad():

    for images, labels in test_loader:

        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)

        _, predicted = torch.max(outputs,1)

        all_labels.extend(labels.cpu().numpy())

        all_predictions.extend(predicted.cpu().numpy())

cm = confusion_matrix(all_labels, all_predictions)

plt.figure(figsize=(8,6))

sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=train_dataset.classes,
    yticklabels=train_dataset.classes
)

plt.xlabel("Predicted")

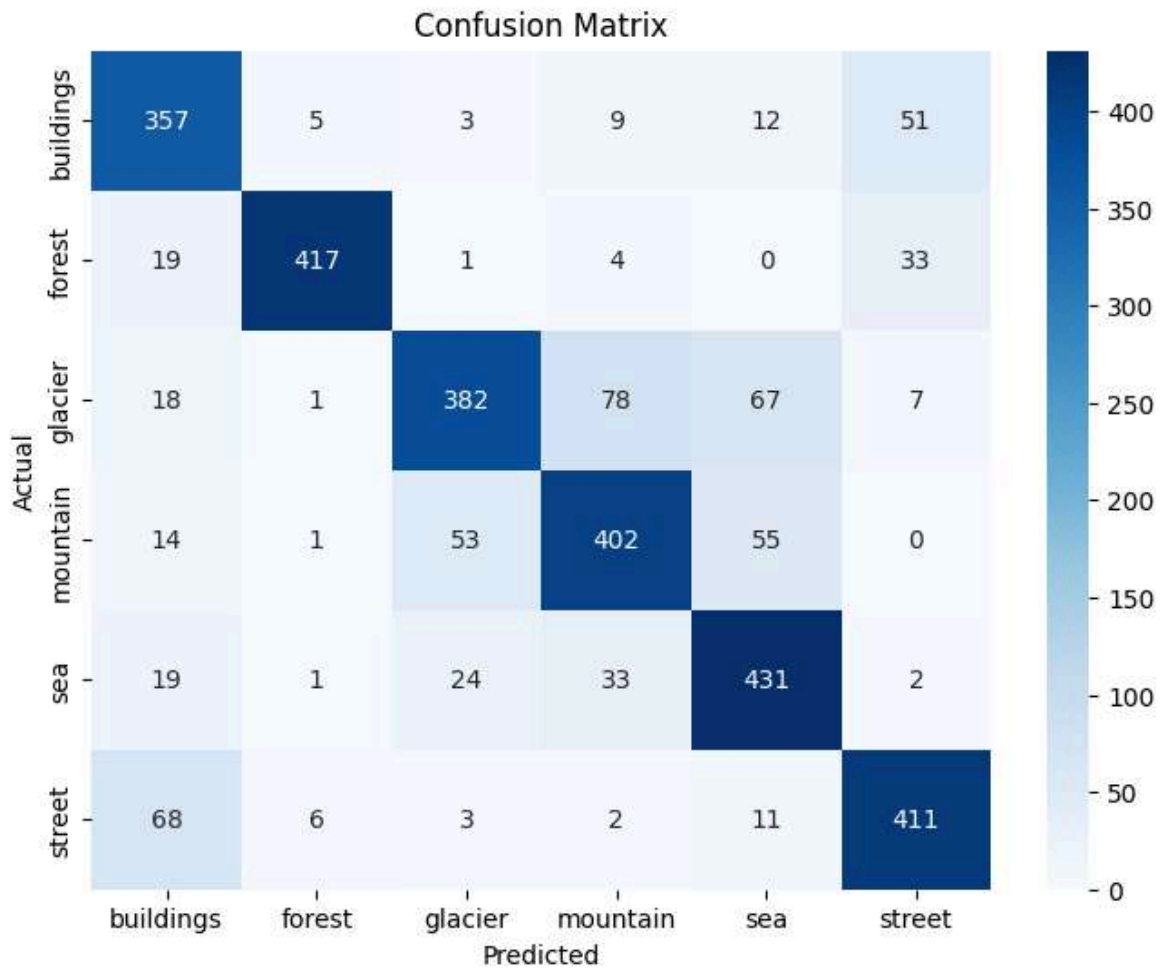
plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

```

Confusion Matrix



In [33]: `from sklearn.metrics import classification_report`

```
print(
    classification_report(
        all_labels,
        all_predictions,
        target_names=train_dataset.classes
    )
)
```

	precision	recall	f1-score	support
buildings	0.72	0.82	0.77	437
forest	0.97	0.88	0.92	474
glacier	0.82	0.69	0.75	553
mountain	0.76	0.77	0.76	525
sea	0.75	0.85	0.79	510
street	0.82	0.82	0.82	501
accuracy			0.80	3000
macro avg	0.81	0.80	0.80	3000
weighted avg	0.81	0.80	0.80	3000

In [34]: `torch.save(
 model.state_dict(),
 "cnn_model.pth"
)`

`print("Model Saved Successfully")`

Model Saved Successfully

```
In [35]: for epoch in range(num_epochs):

    model.train()

    running_loss = 0
    correct = 0
    total = 0

    for images, labels in train_loader:

        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(images)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running_loss += loss.item()

        _, predicted = torch.max(outputs, 1)

        total += labels.size(0)

        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    epoch_acc = 100 * correct / total

    print(
        f"Epoch {epoch+1}/{num_epochs} | Loss: {epoch_loss:.4f} | Accuracy: {epoch_acc:.2f}%"
    )
```

```
Epoch 1/5 | Loss: 0.5792 | Accuracy: 79.42%
Epoch 2/5 | Loss: 0.5434 | Accuracy: 80.40%
Epoch 3/5 | Loss: 0.5263 | Accuracy: 81.54%
Epoch 4/5 | Loss: 0.5019 | Accuracy: 82.19%
Epoch 5/5 | Loss: 0.4788 | Accuracy: 83.16%
```

```
In [36]: print ('تم انتهاء الجزء الاول من المشروع الحمد لله')
```

تم انتهاء الجزء الاول من المشروع الحمد لله

```
In [ ]:
```