```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/771.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/48.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/61.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/377.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/348.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/148.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/177.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/407.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/376.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/162.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/766.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/75.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/89.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/164.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/605.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/411.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/759.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/362.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/176.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/76.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/163.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/229.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/604.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/200.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/607.png
```

```
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/405.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/49.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/599.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/161.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/798.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/360.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/610.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/767.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/372.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/215.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/414.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/439.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/412.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/809.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/770.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/389.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/349.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/598.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/404.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/174.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/808.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/570.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/764.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/214.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/639.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/572.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/63.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/202.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/758.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/602.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/799.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/559.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/612.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/216.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/374.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/366.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/188.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/77.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/638.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/361.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/567.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/573.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/74.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/363.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/772.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/571.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/160.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/406.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/438.png
```

```
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/375.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/203.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/88.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/400.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/606.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/399.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/189.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/410.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/564.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/565.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/60.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/765.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/228.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/201.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/413.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/773.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/149.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/613.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/566.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/558.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/611.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/62.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/388.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/175.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/428.png
/kaggle/input/ultrasound-img/ultrasound_img/non-ds/217.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/709.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/278.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/407.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/119.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/322.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/163.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/345.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/147.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/85.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/102.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/533.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/520.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/288.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/316.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/610.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/370.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/473.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/82.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/152.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/712.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/735.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/166.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/795.png
```

```
/kaggle/input/ultrasound-img/ultrasound_img/ds/86.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/118.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/179.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/793.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/172.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/406.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/643.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/45.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/400.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/44.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/516.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/531.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/183.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/134.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/285.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/62.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/0.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/19.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/257.png
/kaggle/input/ultrasound-img/ultrasound_img/ds/518.png
```

```
import os
import torch
import torchvision.transforms as transforms
from torchvision import datasets, models
from torch import nn, optim
from torch.utils.data import DataLoader, random_split

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define data transformations
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]),
])

# Load Down syndrome dataset
down_syndrome_dataset =
datasets.ImageFolder(root="/kaggle/input/ultrasound-img/ultrasound_img
", transform=transform)

# Load normal dataset
normal_dataset = datasets.ImageFolder(root="/kaggle/input/ultrasound-
img/ultrasound_img", transform=transform)

# Concatenate datasets
full_dataset = torch.utils.data.ConcatDataset([down_syndrome_dataset,
```

```
normal_dataset])

# Split dataset into train and validation sets
train_size = int(0.8 * len(full_dataset))  # 80% training, 20%
validation
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = random_split(full_dataset, [train_size,
val_size])

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=False)

# Load the pre-trained VGG model
model = models.vgg16(pretrained=True)

# Freeze all layers
for param in model.parameters():
    param.requires_grad = False

# Replace the classifier with a new one, suitable for binary
classification
model.classifier = nn.Sequential(
    nn.Linear(25088, 4096),  # Modify the input size based on the
features extracted
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Linear(4096, 2)  # Output 2 classes: Down Syndrome and Normal
)

# Move the model to the device
model = model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

```
        running_loss += loss.item()

    print(f'Epoch [{epoch+1}/{num_epochs}], Training Loss:
{running_loss/len(train_loader):.4f}')

    # Validation loop
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Epoch [{epoch+1}/{num_epochs}], Validation Loss:
{val_loss/len(val_loader):.4f}, Validation Accuracy: {(100 * correct /
total):.2f}%')

# Save the trained model
model_save_path = '/kaggle/working/custom_down_syndrome_model.pth'
torch.save(model.state_dict(), model_save_path)
```

```
/opt/conda/lib/python3.10/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(
/opt/conda/lib/python3.10/site-packages/torchvision/models/_utils.py:2
23: 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=VGG16_Weights.IMAGENET1K_V1`. You can also use
`weights=VGG16_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth"
to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%|██████████| 528M/528M [00:03<00:00, 172MB/s]

Epoch [1/10], Training Loss: 3.9109
Epoch [1/10], Validation Loss: 1.9914, Validation Accuracy: 82.76%
Epoch [2/10], Training Loss: 0.9556
Epoch [2/10], Validation Loss: 1.6282, Validation Accuracy: 86.21%
```

```
Epoch [3/10], Training Loss: 1.1156
Epoch [3/10], Validation Loss: 0.5324, Validation Accuracy: 93.10%
Epoch [4/10], Training Loss: 0.1715
Epoch [4/10], Validation Loss: 0.1726, Validation Accuracy: 96.55%
Epoch [5/10], Training Loss: 0.5039
Epoch [5/10], Validation Loss: 0.2165, Validation Accuracy: 96.55%
Epoch [6/10], Training Loss: 0.1169
Epoch [6/10], Validation Loss: 1.0462, Validation Accuracy: 93.10%
Epoch [7/10], Training Loss: 0.3069
Epoch [7/10], Validation Loss: 0.4542, Validation Accuracy: 96.55%
Epoch [8/10], Training Loss: 0.2933
Epoch [8/10], Validation Loss: 0.0000, Validation Accuracy: 100.00%
Epoch [9/10], Training Loss: 0.7981
Epoch [9/10], Validation Loss: 0.3194, Validation Accuracy: 98.28%
Epoch [10/10], Training Loss: 0.7097
Epoch [10/10], Validation Loss: 0.7456, Validation Accuracy: 94.83%
```

```python
import torch
import torchvision.transforms as transforms
from torchvision import models
from PIL import Image

# Define device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Define data transformations for prediction
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]),
])

# Function to load the model
def load_model(model_path):
    # Load a pre-trained VGG model
    model = models.vgg16(pretrained=True)

    # Replace the classifier with a new one, suitable for binary
classification
    model.classifier = nn.Sequential(
        nn.Linear(25088, 4096),  # Modify the input size based on the
features extracted
        nn.ReLU(inplace=True),
        nn.Dropout(p=0.5),
        nn.Linear(4096, 2)  # Output 2 classes: Down Syndrome and
Normal
    )

    # Load the trained weights
```

```
    model.load_state_dict(torch.load(model_path, map_location=device))

    # Set the model to evaluation mode
    model.eval()

    # Move model to device
    model = model.to(device)

    return model

# Example usage to load the model
model_path = '/kaggle/working/custom_down_syndrome_model.pth'  #
Adjust path if necessary
model = load_model(model_path)

# Function to predict
def predict(image_path):
    # Load and transform the image
    img = Image.open(image_path)
    img = transform(img).unsqueeze(0).to(device)

    # Predict the class
    with torch.no_grad():
        outputs = model(img)
        _, predicted = torch.max(outputs, 1)

        # Return the prediction label
        return 'Down Syndrome' if predicted.item() == 0 else 'Normal'

# Example usage of the predict function
image_path = '/kaggle/input/ultrasound-img/ultrasound_img/ds/163.png'
# Replace with the path to your test image
prediction = predict(image_path)
print(f'Prediction: {prediction}')
```

```
Prediction: Down Syndrome
```

```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

def evaluate_vgg16(predictions, ground_truths):
    # Convert predictions and ground truths to numpy arrays
    predictions = np.array(predictions)
    ground_truths = np.array(ground_truths)

    # Calculate metrics
    accuracy = accuracy_score(ground_truths, predictions)
    precision = precision_score(ground_truths, predictions,
average='binary')
```

```
    recall = recall_score(ground_truths, predictions,
average='binary')
    f1 = f1_score(ground_truths, predictions, average='binary')
    conf_matrix = confusion_matrix(ground_truths, predictions)

    return accuracy, precision, recall, f1, conf_matrix

# Example usage
predictions = [0, 1, 0, 1, 1]  # Example predictions
ground_truths = [0, 1, 1, 1, 0]  # Example ground truths
accuracy, precision, recall, f1, conf_matrix =
evaluate_vgg16(predictions, ground_truths)
print(f'VGG-16 Accuracy: {accuracy}')
print(f'VGG-16 Precision: {precision}')
print(f'VGG-16 Recall: {recall}')
print(f'VGG-16 F1 Score: {f1}')
print(f'Confusion Matrix:\n{conf_matrix}')
```

```
VGG-16 Accuracy: 0.6
VGG-16 Precision: 0.6666666666666666
VGG-16 Recall: 0.6666666666666666
VGG-16 F1 Score: 0.6666666666666666
Confusion Matrix:
[[1 1]
 [1 2]]
```