```
!cat kaggle.json
```

```
{"username":"amaritanshigupta","key":"7d8734f14c1e017827989f412e749a5d"}
```

```
!pip install -q kaggle

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

```
!kaggle --version
```

```
Kaggle API 1.7.4.5
```

```
!kaggle datasets download -d jtiptj/chest-xray-pneumoniacovid19tuberculosis
```

```
!ls
```

```
!unzip chest-xray-pneumoniacovid19tuberculosis.zip
```

```
  inflating: train/PNEUMONIA/person727_virus_1347.jpeg
  inflating: train/PNEUMONIA/person728_bacteria_2630.jpeg
  inflating: train/PNEUMONIA/person72_bacteria_352.jpeg
  inflating: train/PNEUMONIA/person72_bacteria_353.jpeg
  inflating: train/PNEUMONIA/person72_bacteria_354.jpeg
  inflating: train/PNEUMONIA/person730_bacteria_2632.jpeg
  inflating: train/PNEUMONIA/person730_virus_1351.jpeg
  inflating: train/PNEUMONIA/person731_bacteria_2633.jpeg
  inflating: train/PNEUMONIA/person731_virus_1352.jpeg
  inflating: train/PNEUMONIA/person732_bacteria_2634.jpeg
  inflating: train/PNEUMONIA/person732_virus_1353.jpeg
  inflating: train/PNEUMONIA/person733_bacteria_2635.jpeg
  inflating: train/PNEUMONIA/person734_bacteria_2637.jpeg
  inflating: train/PNEUMONIA/person734_virus_1355.jpeg
```

```
!ls
```

```
chest-xray-pneumoniacovid19tuberculosis.zip  sample_data  train
kaggle.json                                  test         val
```

```
!mv train/TURBERCULOSIS train/TUBERCULOSIS
!mv val/TURBERCULOSIS val/TUBERCULOSIS
!mv test/TURBERCULOSIS test/TUBERCULOSIS
```

```
!ls train
!ls val
!ls test
```

```
COVID19  NORMAL  PNEUMONIA  TUBERCULOSIS
COVID19  NORMAL  PNEUMONIA  TUBERCULOSIS
COVID19  NORMAL  PNEUMONIA  TUBERCULOSIS
```

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

train_tfms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

eval_tfms = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225])
])

train_ds = datasets.ImageFolder("/content/train", transform=train_tfms)
val_ds   = datasets.ImageFolder("/content/val",   transform=eval_tfms)
test_ds  = datasets.ImageFolder("/content/test",  transform=eval_tfms)

train_loader = DataLoader(train_ds, batch_size=16, shuffle=True, num_workers=0)
val_loader   = DataLoader(val_ds,   batch_size=16, num_workers=0)
test_loader  = DataLoader(test_ds,  batch_size=16, num_workers=0)

print("Classes:", train_ds.class_to_idx)
```

```
Classes: {'COVID19': 0, 'NORMAL': 1, 'PNEUMONIA': 2, 'TUBERCULOSIS': 3}
```

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import timm
import torch.nn as nn
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
```

```
Device: cuda
```

```
!ls /content
```

```
chest-xray-pneumoniacovid19tuberculosis.zip  sample_data  train
kaggle.json                                  test         val
```

```
def train_epoch():
    model.train()
    total = 0
    for x,y in train_loader:
        x,y = x.to(device), y.to(device)
        optimizer.zero_grad()
        out = model(x)
        loss = criterion(out,y)
        loss.backward()
        optimizer.step()
        total += loss.item()
    return total/len(train_loader)

def eval_epoch(loader):
    model.eval()
    correct=total=0
    with torch.no_grad():
        for x,y in loader:
            x,y = x.to(device), y.to(device)
            _,p = model(x).max(1)
            total += y.size(0)
        correct += (p==y).sum().item()
    return correct/total
```

```
model = timm.create_model(
    "densenet121",
    pretrained=True,
    num_classes=4
)

model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

print("Model defined and moved to device")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104: UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret HF_TOKEN timed out. Secrets
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub (https://github.com/huggingface/hugg
  warnings.warn(
model.safetensors:   0%|          | 0.00/32.3M [00:00<?, ?B/s]
Model defined and moved to device
```

```
print(type(model))
```

```
<class 'timm.models.densenet.DenseNet'>
```

```
best = 0.0
```

```
for epoch in range(1, 11):
    model.train()
    running_loss = 0.0

    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()

    val_acc = eval_epoch(val_loader)
    print(f"Epoch {epoch} | Loss {running_loss/len(train_loader):.4f} | Val Acc {val_acc:.4f}")

    if val_acc > best:
        best = val_acc
        torch.save(model.state_dict(), "best_densenet.pth")
```

```
Epoch 1 | Loss 0.2501 | Val Acc 0.7368
Epoch 2 | Loss 0.0917 | Val Acc 0.7105
Epoch 3 | Loss 0.0639 | Val Acc 0.8684
Epoch 4 | Loss 0.0515 | Val Acc 0.8947
Epoch 5 | Loss 0.0418 | Val Acc 0.8421
Epoch 6 | Loss 0.0333 | Val Acc 0.8421
Epoch 7 | Loss 0.0275 | Val Acc 0.8421
Epoch 8 | Loss 0.0271 | Val Acc 0.8947
Epoch 9 | Loss 0.0202 | Val Acc 0.8947
Epoch 10 | Loss 0.0190 | Val Acc 0.9211
```

```
model.load_state_dict(torch.load("best_densenet.pth"))
test_acc = eval_epoch(test_loader)
print("Final Test Accuracy:", test_acc)
```

```
Final Test Accuracy: 0.9040207522697795
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class CustomCNN(nn.Module):
    def __init__(self, num_classes=4):
        super(CustomCNN, self).__init__()

        # Block 1
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1   = nn.BatchNorm2d(32)

        # Block 2
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2   = nn.BatchNorm2d(64)

        # Block 3
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3   = nn.BatchNorm2d(128)

        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)

        # After 3 pools: 224 → 112 → 56 → 28
        self.fc1 = nn.Linear(128 * 28 * 28, 256)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))

        x = x.view(x.size(0), -1)
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)

        return x
```

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

model = CustomCNN(num_classes=4).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

print(model)
```

```
CustomCNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=100352, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=4, bias=True)
)
```

```
def train_epoch():
    model.train()
    total_loss = 0.0

    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()

        total_loss += loss.item()

    return total_loss / len(train_loader)


def eval_epoch(loader):
    model.eval()
    correct = total = 0

    with torch.no_grad():
        for images, labels in loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.max(outputs, 1)

            total += labels.size(0)
            correct += (preds == labels).sum().item()
    return correct / total
```

```
best_acc = 0.0

for epoch in range(1, 16):
    loss = train_epoch()
    val_acc = eval_epoch(val_loader)

    print(f"Epoch {epoch} | Loss: {loss:.4f} | Val Acc: {val_acc:.4f}")

    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(model.state_dict(), "best_custom_cnn.pth")
```

```
Epoch 1 | Loss: 0.1098 | Val Acc: 0.8947
Epoch 2 | Loss: 0.1176 | Val Acc: 0.8158
Epoch 3 | Loss: 0.1075 | Val Acc: 0.8421
Epoch 4 | Loss: 0.1106 | Val Acc: 0.8421
Epoch 5 | Loss: 0.1158 | Val Acc: 0.7368
Epoch 6 | Loss: 0.1124 | Val Acc: 0.9474
Epoch 7 | Loss: 0.1087 | Val Acc: 0.9211
Epoch 8 | Loss: 0.1063 | Val Acc: 0.8421
Epoch 9 | Loss: 0.1026 | Val Acc: 0.9474
Epoch 10 | Loss: 0.0936 | Val Acc: 0.7368
Epoch 11 | Loss: 0.0876 | Val Acc: 0.8158
Epoch 12 | Loss: 0.0884 | Val Acc: 0.8947
Epoch 13 | Loss: 0.0942 | Val Acc: 0.7632
Epoch 14 | Loss: 0.0853 | Val Acc: 0.9474
Epoch 15 | Loss: 0.0854 | Val Acc: 0.8158
```

```
model.load_state_dict(torch.load("best_custom_cnn.pth"))
test_acc = eval_epoch(test_loader)
print("Custom CNN Test Accuracy:", test_acc)
```

```
Custom CNN Test Accuracy: 0.8223086900129701
```

```
import torch

def evaluate_model(model, loader):
    model.eval()
    correct = total = 0

    with torch.no_grad():
        for images, labels in loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, preds = torch.max(outputs, 1)

            total += labels.size(0)
            correct += (preds == labels).sum().item()

    return correct / total
```

```
# Load DenseNet
densenet = timm.create_model(
    "densenet121",
    pretrained=False,
    num_classes=4
).to(device)

densenet.load_state_dict(torch.load("best_densenet.pth"))

# Load Custom CNN
custom_cnn = CustomCNN(num_classes=4).to(device)
```

```
custom_cnn.load_state_dict(torch.load("best_custom_cnn.pth"))

# Evaluate
densenet_acc = evaluate_model(densenet, test_loader)
cnn_acc = evaluate_model(custom_cnn, test_loader)

print(f"DenseNet Test Accuracy: {densenet_acc:.4f}")
print(f"Custom CNN Test Accuracy: {cnn_acc:.4f}")
```

```
DenseNet Test Accuracy: 0.9040
Custom CNN Test Accuracy: 0.8223
```

```
best_model = densenet if densenet_acc >= cnn_acc else custom_cnn
best_model_name = "DenseNet-121" if densenet_acc >= cnn_acc else "Custom CNN"

print("Best Model:", best_model_name)
```

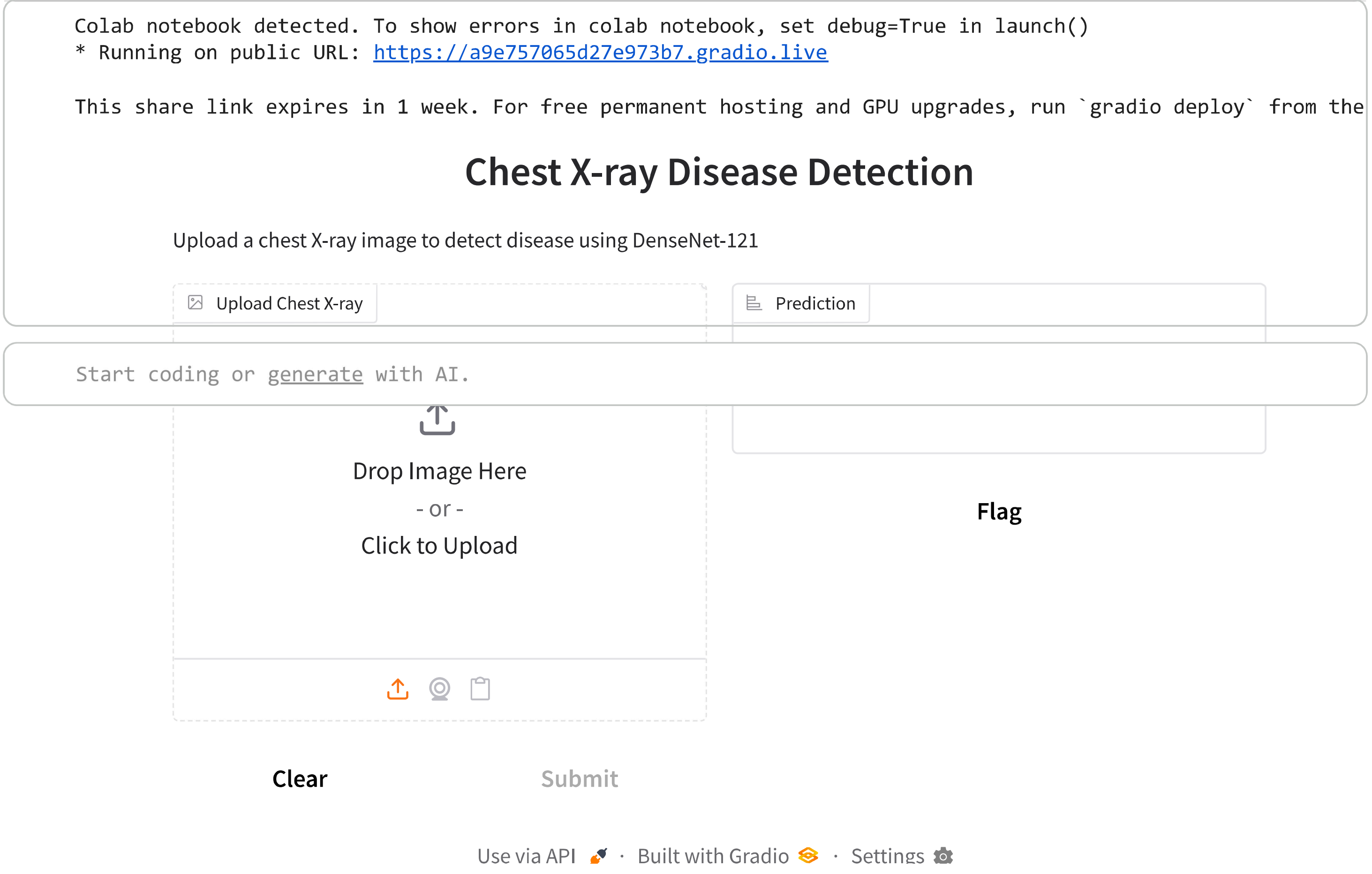
```
Best Model: DenseNet-121
```

```
torch.save(best_model.state_dict(), "final_best_model.pth")
```

```
!pip install -q gradio pillow
```

```
!pip install -q gradio pillow opencv-python
```

```
import torch
import timm
import gradio as gr
from PIL import Image
from torchvision import transforms

# Classes
CLASSES = ["COVID19", "NORMAL", "PNEUMONIA", "TUBERCULOSIS"]

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

# Load best model (DenseNet)
model = timm.create_model("densenet121", pretrained=False, num_classes=4)
model.load_state_dict(torch.load("final_best_model.pth", map_location=device))
model.to(device)
model.eval()

# Image transform
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]
    )
])

def predict_xray(image):
    image = image.convert("RGB")
    img = transform(image).unsqueeze(0).to(device)

    with torch.no_grad():
        outputs = model(img)
        probs = torch.softmax(outputs, dim=1)
        pred = torch.argmax(probs, dim=1).item()

    return {
        CLASSES[i]: float(probs[0][i])
        for i in range(len(CLASSES))
    }

# Gradio Interface
interface = gr.Interface(
    fn=predict_xray,
    inputs=gr.Image(type="pil", label="Upload Chest X-ray"),
    outputs=gr.Label(num_top_classes=4, label="Prediction"),
    title="Chest X-ray Disease Detection",
    description="Upload a chest X-ray image to detect disease using DenseNet-121"
)

interface.launch(share=True)
```



```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://a9e757065d27e973b7.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the
```

Chest X-ray Disease Detection

Upload a chest X-ray image to detect disease using DenseNet-121