```
from google.colab import files
files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"aryanmittal07","key":"22e5d1c54ba5e16f3fa1353a13a1e56a"}'}
```

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

```
mv: cannot stat 'kaggle.json': No such file or directory
```

```
!kaggle datasets list
```

```
ref                                                                 title                                             size  lastUpdated                 downloadCoun
------------------------------------------------------------------  ------------------------------------------------  ----------  --------------------------  ------------
saidaminsaidaxmadov/chocolate-sales                                 Chocolate Sales                                   468320  2026-01-04 14:23:35.490000
rockyt07/social-media-user-analysis                                 Social Media User Analysis                      247842357  2026-01-14 02:28:41.970000
neurocipher/heartdisease                                            Heart Disease                                     3491  2025-12-11 15:29:14.327000
vishardmehta/indian-engineering-college-placement-dataset           Indian Engineering College Placement Dataset    137603  2026-01-24 15:23:40.150000
thedrzee/student-social-media-and-relationships                     Student Social Media & Relationships              7851  2026-01-20 19:00:03.917000
ayeshasiddiqa123/student-perfirmance                                Student Academic Performance Dataset.            96178  2026-01-06 12:08:32.540000
nudratabbas/vitamin-deficiency-disease-prediction-dataset           Vitamin Deficiency Disease Prediction Dataset    169913  2026-01-23 07:02:35.970000
algozee/bmw-dataset                                                 BMW Used Car Market Analysis Dataset             112601  2026-01-13 16:50:57.680000
dhrubangtalukdar/qs-world-university-rankings-2026-top-1500         QS World University Rankings 2026 - Top 1500      79192  2026-01-23 11:11:49.313000
hassanjameelahmed/healthcare-analytics-patient-flow-data            healthcare_analytics_patient_flow_data           234313  2026-01-22 12:32:26.693000
dhrubangtalukdar/telco-customer-churn-data                          Telco Customer Churn Data                       1507263  2026-01-24 23:17:31.177000
mohamedasak/metacritic-movies-dataset                               Metacritic Movies Dataset                       9257612  2026-01-24 15:13:42.403000
shraddha4ever20/top-rated-movies-from-tmdb-19902025                 Top Rated Movies from TMDb (1902-2026)          2651890  2026-01-22 11:56:53.660000
neurocipher/student-performance                                     Student Performance                               49705  2025-12-12 12:06:28.973000
sarcasmos/ai-society                                                Global AI Impact on Jobs (2010-2025)             306774  2026-01-19 08:44:29.597000
muhammadaammartufail/silver-prices-10-year-data-and-2026-forecast   Silver Prices: 10-Year Data & 2026 Forecast       42403  2026-01-16 21:24:19.647000
hassanjameelahmed/alibaba-baba-stock-market-dataset-20142023        Alibaba (BABA) Stock Market Dataset (2014-2023)   65301  2026-01-21 08:12:21.497000
ishank2005/salary-csv                                               Salary.csv                                          392  2025-12-29 15:48:58.240000
dhrubangtalukdar/afcon-202526-complete-match-statistics             ⭐ AFCON 2025-26 Complete Match Statistics         3396  2026-01-22 09:09:02.060000
aliiihussain/daily-global-stock-market-indicators                   Daily Global Stock Market Indicators             507856  2026-01-18 20:33:55.790000
```

```
!pip install -q kaggle
```

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

```
Dataset URL: https://www.kaggle.com/datasets/jtiptj/chest-xray-pneumoniacovid19tuberculosis
License(s): other
Downloading chest-xray-pneumoniacovid19tuberculosis.zip to /content
 96% 1.68G/1.74G [00:19<00:01, 45.8MB/s]
100% 1.74G/1.74G [00:19<00:00, 94.4MB/s]
```

```
!ls
```

```
chest-xray-pneumoniacovid19tuberculosis.zip  sample_data
```

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

```
  inflating: train/NORMAL/NORMAL2-IM-1067-0001-0001.jpeg
```

```
  inflating: train/NORMAL/NORMAL2-IM-1067-0001.jpeg
```

```
!ls
```

```
chest-xray-pneumoniacovid19tuberculosis.zip  sample_data  test  train  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  test  train  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:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and resta
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model.safetensors:  100%                                                32.3M/32.3M  [00:01<00:00,  29.6MB/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 0.2555 | Val Acc 0.8684
Epoch 2 | Loss 0.0979 | Val Acc 0.9737
Epoch 3 | Loss 0.0650 | Val Acc 0.8421
Epoch 4 | Loss 0.0527 | Val Acc 0.9211
Epoch 5 | Loss 0.0453 | Val Acc 0.8158
Epoch 6 | Loss 0.0367 | Val Acc 0.9474
Epoch 7 | Loss 0.0247 | Val Acc 0.8158
Epoch 8 | Loss 0.0227 | Val Acc 0.8947
Epoch 9 | Loss 0.0232 | Val Acc 0.9737
Epoch 10 | Loss 0.0207 | Val Acc 0.8684
```

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

```
Final Test Accuracy: 0.920881971465629
```

```
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: 0.4534 | Val Acc: 0.6579
Epoch 2 | Loss: 0.2769 | Val Acc: 0.8947
Epoch 3 | Loss: 0.2193 | Val Acc: 0.8158
Epoch 4 | Loss: 0.1958 | Val Acc: 0.8947
Epoch 5 | Loss: 0.1871 | Val Acc: 0.8684
Epoch 6 | Loss: 0.1708 | Val Acc: 0.7895
Epoch 7 | Loss: 0.1638 | Val Acc: 0.7105
Epoch 8 | Loss: 0.1614 | Val Acc: 0.8421
Epoch 9 | Loss: 0.1439 | Val Acc: 0.8158
Epoch 10 | Loss: 0.1387 | Val Acc: 0.8947
Epoch 11 | Loss: 0.1442 | Val Acc: 0.8158
Epoch 12 | Loss: 0.1396 | Val Acc: 0.8684
Epoch 13 | Loss: 0.1220 | Val Acc: 0.8158
Epoch 14 | Loss: 0.1141 | Val Acc: 0.8421
Epoch 15 | Loss: 0.1109 | Val Acc: 0.9211
```

```
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: 0.8015564202334631
```

```
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: 0.9209
Custom CNN Test Accuracy: 0.8016
```

```
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
import numpy as np
import cv2
from PIL import Image
from torchvision import transforms

# ---------------- CONFIG ----------------
CLASSES = ["COVID19", "NORMAL", "PNEUMONIA", "TUBERCULOSIS"]
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ---------------- LOAD MODEL ----------------
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()

# ---------------- TRANSFORM ----------------
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485,0.456,0.406],
                         [0.229,0.224,0.225])
])

# ---------------- GRAD-CAM FIXED ----------------
class GradCAM:
    def __init__(self, model, target_layer):
```

```
        def __init__(self, model, target_layer):
            self.model = model
            self.gradients = None
            self.activations = None

            target_layer.register_forward_hook(self.save_activation)
            target_layer.register_full_backward_hook(self.save_gradient)

        def save_activation(self, module, inp, out):
            self.activations = out.detach()

        def save_gradient(self, module, grad_input, grad_output):
            self.gradients = grad_output[0].detach()

        def generate(self):
            weights = self.gradients.mean(dim=(2,3), keepdim=True)
            cam = (weights * self.activations).sum(dim=1)
            cam = torch.relu(cam)
            cam = cam[0].cpu().numpy()
            cam = (cam - cam.min()) / (cam.max() - cam.min() + 1e-8)
            return cam

    # Last conv layer of DenseNet
    gradcam = GradCAM(model, model.features[-1])

    # ---------------- PREDICT FUNCTION ----------------
    def predict_xray(image):
        try:
            image = image.convert("RGB")
            orig = np.array(image.resize((224,224)))

            img = transform(image).unsqueeze(0).to(device)
            output = model(img)
            probs = torch.softmax(output, dim=1)

            pred = torch.argmax(probs, dim=1).item()

            model.zero_grad()
            output[0, pred].backward()

            cam = gradcam.generate()
            cam = cv2.resize(cam, (224,224))
            heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)
            overlay = cv2.addWeighted(orig, 0.6, heatmap, 0.4, 0)

            return (
                CLASSES[pred],
                f"{probs[0][pred].item()*100:.2f}%",
                Image.fromarray(overlay)
            )

        except Exception as e:
            return "Error", "Error", None

    # ---------------- GUI (UPDATED) ----------------
    with gr.Blocks(theme=gr.themes.Soft()) as demo:
        gr.Markdown(
```

```
        """
        # 🩺 Chest X-ray Disease Detection (Explainable AI)
        Upload a chest X-ray image to detect the disease and visualize
        **infected lung regions using Grad-CAM**.
        """
    )

    with gr.Row():
        with gr.Column(scale=1):
            xray_input = gr.Image(
                type="pil",
                label="Upload Chest X-ray",
                height=300
            )
            predict_btn = gr.Button("🔍 Analyze X-ray")

        with gr.Column(scale=1):
            disease_output = gr.Textbox(
                label="Predicted Disease",
                interactive=False
            )
            confidence_output = gr.Textbox(
                label="Confidence",
                interactive=False
            )
            heatmap_output = gr.Image(
                label="Grad-CAM Heatmap",
                height=300
            )

    predict_btn.click(
        fn=predict_xray,
        inputs=xray_input,
        outputs=[disease_output, confidence_output, heatmap_output]
    )

demo.launch(share=True)
```

```
/tmp/ipython-input-3570812511.py:84: DeprecationWarning: The 'theme' parameter in the Blocks constructor will be removed in Gradio 6.0. You will need to pass 'theme' to
  with gr.Blocks(theme=gr.themes.Soft()) as demo:
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://c2ea5ca5b163821cb7.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face
```

No interface is running right now

Start coding or generate with AI.