Multi-Class COVID-19 Detection from Chest X-Ray Images

Complete Technical Documentation

Author: Sachin Dattatraay Mosambe **Project**: AI-Powered Medical Image Classification **Deployed App**: http://3.7.46.231:8501/

Table of Contents

Project Overview

Objective

Build an AI diagnostic system to classify chest X-ray images into:

- **COVID-19**
- **Viral Pneumonia**
- **Normal**

Key Features

☑ Multi-class classification with 94% accuracy ☑ Transfer learning with DenseNet-121 ☑ Grad-CAM visual explanations ☑ Real-time web interface (Streamlit) ☑ REST API (FastAPI) ☑ AWS deployment (EC2 + S3) ☑ CI/CD with GitHub Actions

Technical Stack

Component	Technology
Framework	PyTorch 2.3.0
Model	DenseNet-121
Explainability	Grad-CAM

Component	Technology
Web App	Streamlit
API	FastAPI
Cloud	AWS EC2 + S3
CI/CD	GitHub Actions

Data Pipeline

Dataset Structure

```
Covid19-dataset/
├── train/
│   ├── Covid/              (~137 images)
│   ├── Normal/             (~200 images)
│   └── Viral Pneumonia/    (~200 images)
└── test/                    (66 images total)
```

Data Preprocessing

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

# Image transformations
IMG_SIZE = 224
BATCH_SIZE = 16

# Training: with augmentation
train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Testing: no augmentation
test_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# Load datasets
train_dataset = datasets.ImageFolder("Covid19-dataset/train", train_transform)
test_dataset = datasets.ImageFolder("Covid19-dataset/test", test_transform)
```

```
# Split train into train/val (80/20)
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_set, val_set = random_split(train_dataset, [train_size, val_size])

# Create data loaders
train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_set, batch_size=BATCH_SIZE, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

class_names = train_dataset.classes  # ['Covid', 'Normal', 'Viral Pneumonia']
```

Model Development

Model Comparison Results

Model	Test Accuracy	COVID Precision	COVID Recall	ROC-AUC
SimpleCNN	0.8750	0.9231	0.9231	0.9286
ResNet-50	**1.0000**	1.0000	1.0000	1.0000
VGG-16	0.8333	1.0000	1.0000	0.9667
DenseNet-121	**0.9697**	1.0000	1.0000	0.9917

DenseNet-121 Implementation

```
import torch
import torch.nn as nn
from torchvision import models

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

# Load pre-trained DenseNet-121
model = models.densenet121(pretrained=True)

# Modify classifier for 3 classes
num_classes = 3
model.classifier = nn.Linear(model.classifier.in_features, num_classes)
model = model.to(device)

print(f"Total parameters: {sum(p.numel() for p in model.parameters()):,}")
```

Training Configuration

```
import torch.optim as optim
```

```
# Hyperparameters
EPOCHS = 15
LR = 0.0001
WEIGHT_DECAY = 0.01

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=EPOCHS)
```

Training Loop

```
best_acc = 0
history = {'train_loss': [], 'train_acc': [], 'val_loss': [], 'val_acc': []}

for epoch in range(EPOCHS):
    # Training
    model.train()
    train_loss, train_correct, train_total = 0, 0, 0

    for imgs, labels in train_loader:
        imgs, labels = imgs.to(device), labels.to(device)

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

        train_loss += loss.item() * imgs.size(0)
        train_correct += (outputs.argmax(1) == labels).sum().item()
        train_total += labels.size(0)

    train_acc = train_correct / train_total

    # Validation
    model.eval()
    val_correct, val_total = 0, 0

    with torch.no_grad():
        for imgs, labels in val_loader:
            imgs, labels = imgs.to(device), labels.to(device)
            outputs = model(imgs)
            val_correct += (outputs.argmax(1) == labels).sum().item()
            val_total += labels.size(0)

    val_acc = val_correct / val_total

    # Save best model
    if val_acc > best_acc:
        best_acc = val_acc
```

```
            torch.save(model.state_dict(), "best_model.pth")
            print(f"✓ Epoch {epoch+1} | Val Acc: {val_acc:.3f} | SAVED")

        scheduler.step()
```

Model Evaluation

Comprehensive Metrics

```
from sklearn.metrics import (accuracy_score, precision_recall_fscore_support,
                             confusion_matrix, roc_auc_score,
classification_report)
from sklearn.preprocessing import label_binarize

# Load best model
model.load_state_dict(torch.load("best_model.pth"))
model.eval()

# Collect predictions
all_preds, all_labels, all_probs = [], [], []

with torch.no_grad():
    for imgs, labels in test_loader:
        imgs = imgs.to(device)
        outputs = model(imgs)
        probs = torch.softmax(outputs, dim=1)

        all_preds.extend(outputs.argmax(1).cpu().numpy())
        all_labels.extend(labels.numpy())
        all_probs.extend(probs.cpu().numpy())

# Calculate metrics
test_acc = accuracy_score(all_labels, all_preds)
precision, recall, f1, _ = precision_recall_fscore_support(all_labels, all_preds,
average='weighted')

# ROC-AUC
y_test_bin = label_binarize(all_labels, classes=range(num_classes))
roc_auc = roc_auc_score(y_test_bin, all_probs, average='macro', multi_class='ovr')

print(f"Accuracy:  {test_acc:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall:    {recall:.4f}")
print(f"F1-Score:  {f1:.4f}")
print(f"ROC-AUC:   {roc_auc:.4f}")
```

Confusion Matrix

```
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(all_labels, all_preds)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
```

Explainable AI

Grad-CAM Implementation

```
import cv2
import torch

class GradCAM:
    """Gradient-weighted Class Activation Mapping"""

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

        # Hook last convolutional layer
        target_layer = model.features[-1]
        target_layer.register_forward_hook(self.save_activation)
        target_layer.register_full_backward_hook(self.save_gradient)

    def save_activation(self, module, input, output):
        self.activations = output.detach().clone()

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

    def generate_cam(self, input_tensor, target_class=None):
        # Forward pass
        output = self.model(input_tensor)

        if target_class is None:
            target_class = output.argmax(dim=1).item()

        # Backward pass
```

```
        self.model.zero_grad()
        output[0, target_class].backward()

        # Generate CAM
        gradients = self.gradients[0]
        activations = self.activations[0]
        weights = gradients.mean(dim=(1, 2))

        cam = torch.zeros(activations.shape[1:], device=input_tensor.device)
        for i, w in enumerate(weights):
            cam += w * activations[i]

        cam = torch.relu(cam)
        cam = cam.cpu().numpy()
        cam = cv2.resize(cam, (224, 224))
        cam = (cam - cam.min()) / (cam.max() - cam.min() + 1e-8)

        return cam, target_class, output

# Usage
gradcam = GradCAM(model)
cam, pred_class, output = gradcam.generate_cam(input_tensor)
```

Visualization

```
def denormalize_image(tensor):
    img = tensor.squeeze(0).cpu().numpy().transpose(1, 2, 0)
    img = img * [0.229, 0.224, 0.225] + [0.485, 0.456, 0.406]
    return np.clip(img, 0, 1)

def create_overlay(image_np, cam):
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLORMAP_JET)
    heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB) / 255.0
    return heatmap * 0.4 + image_np * 0.6
```

Deployment

Architecture

```
GitHub → GitHub Actions → AWS EC2 → Download Model from S3 → Streamlit App
```

Streamlit Application

```
import streamlit as st
import torch
```

```
from torchvision import models, transforms
from PIL import Image
import boto3

# AWS S3 Configuration
S3_BUCKET_NAME = "bank-term-model-bucket"
S3_MODEL_KEY = "models/model_for_inference.pth"
LOCAL_MODEL_PATH = "/tmp/model_for_inference.pth"

@st.cache_resource
def download_model_from_s3():
    """Download model from S3"""
    if os.path.exists(LOCAL_MODEL_PATH):
        return LOCAL_MODEL_PATH

    s3_client = boto3.client('s3')
    s3_client.download_file(S3_BUCKET_NAME, S3_MODEL_KEY, LOCAL_MODEL_PATH)
    return LOCAL_MODEL_PATH

@st.cache_resource
def load_model():
    model_path = download_model_from_s3()
    checkpoint = torch.load(model_path, map_location='cpu')

    model = models.densenet121(pretrained=False)
    model.classifier = nn.Linear(model.classifier.in_features,
checkpoint["num_classes"])
    model.load_state_dict(checkpoint["model_state_dict"])
    model.eval()

    return model, checkpoint["class_names"]

def main():
    st.title("COVID-19 Detection from Chest X-Ray")

    model, class_names = load_model()

    uploaded_file = st.file_uploader("Upload X-Ray Image", type=["jpg", "png"])

    if uploaded_file:
        image = Image.open(uploaded_file).convert("RGB")

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

        input_tensor = transform(image).unsqueeze(0)

        with torch.no_grad():
            output = model(input_tensor)
            probs = torch.softmax(output, dim=1)[0]
```

```
            pred_class = output.argmax(1).item()

        # Display results
        col1, col2 = st.columns(2)

        with col1:
            st.image(image, caption="Uploaded X-Ray", use_container_width=True)

        with col2:
            st.success(f"**Prediction: {class_names[pred_class]}**")
            st.metric("Confidence", f"{probs[pred_class].item():.1%}")

            st.write("All Predictions:")
            for i, class_name in enumerate(class_names):
                st.write(f"{class_name}: {probs[i].item():.1%}")

if __name__ == "__main__":
    main()
```

FastAPI Implementation

```
from fastapi import FastAPI, File, UploadFile
import uvicorn
from PIL import Image
import io
import torch
from torchvision import models, transforms

app = FastAPI()

# Global model
model = None
class_names = ['Covid', 'Normal', 'Viral Pneumonia']

@app.on_event("startup")
def load_model():
    global model
    model = models.densenet121(pretrained=False)
    model.classifier = torch.nn.Linear(model.classifier.in_features, 3)
    model.load_state_dict(torch.load("best_model.pth", map_location='cpu'))
    model.eval()

@app.post("/predict/")
async def predict(file: UploadFile = File(...)):
    # Read image
    contents = await file.read()
    image = Image.open(io.BytesIO(contents)).convert("RGB")

    # Preprocess
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
```

```
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

    tensor = transform(image).unsqueeze(0)

    # Predict
    with torch.no_grad():
        output = model(tensor)
        probs = torch.nn.functional.softmax(output, dim=1).numpy()[0]

    return {
        "predicted_class": class_names[int(probs.argmax())],
        "probabilities": {
            class_names[i]: float(probs[i]) for i in range(len(class_names))
        }
    }

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

CI/CD Pipeline (GitHub Actions)

```
name: Deploy to EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up SSH
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.EC2_KEY }}" > ~/.ssh/ec2_key.pem
          chmod 600 ~/.ssh/ec2_key.pem

      - name: Deploy to EC2
        run: |
          ssh -o StrictHostKeyChecking=no -i ~/.ssh/ec2_key.pem \
              ${{ secrets.EC2_USER }}@${{ secrets.EC2_HOST }} << 'EOF'
            cd ~/Multi-class-COVID-19-Detection
            git pull origin main
            source venv/bin/activate
```

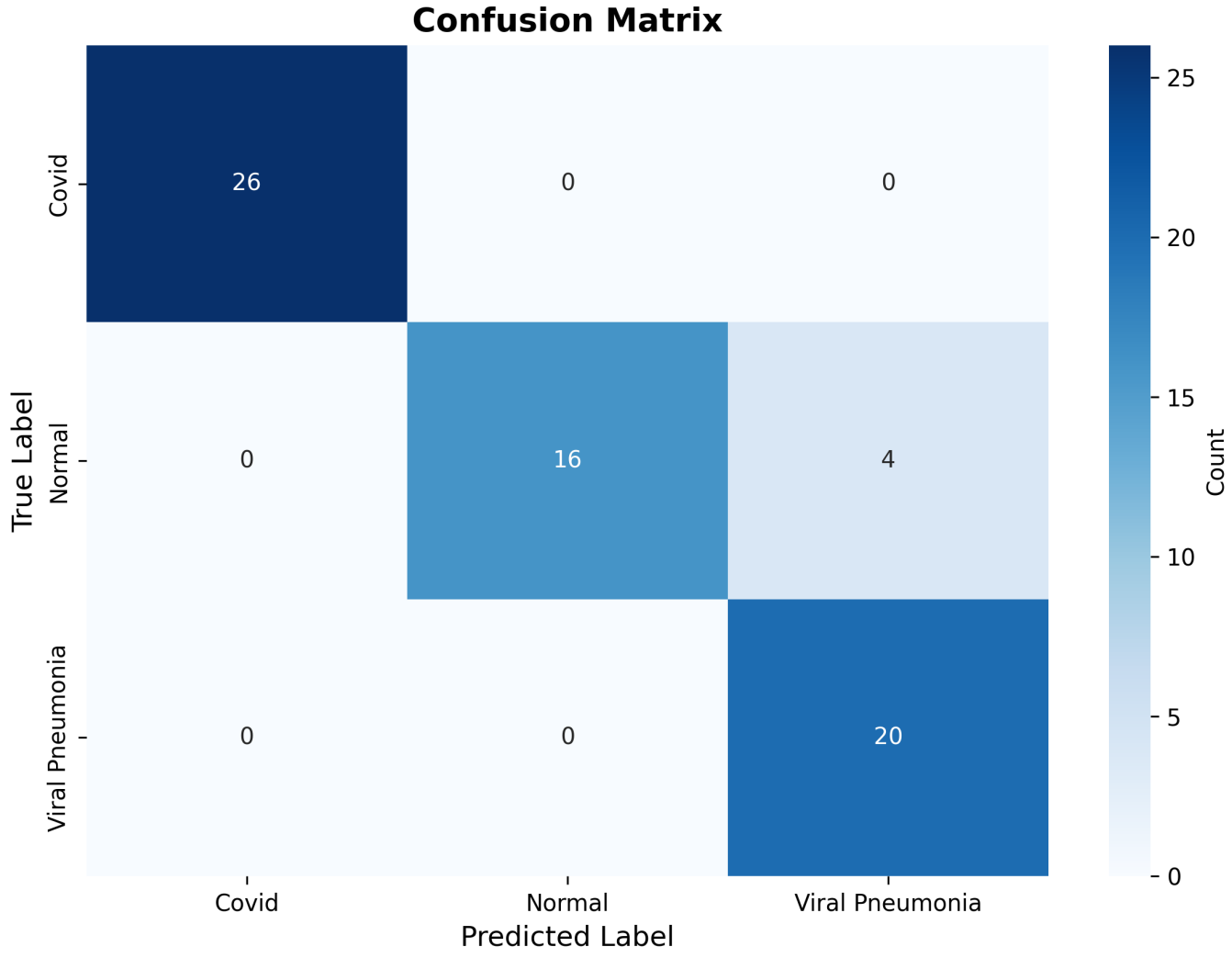
```
            pip install -r requirements.txt

            # Download model from S3
            aws s3 cp s3://${{ secrets.S3_BUCKET }}/models/model_for_inference.pth
\
                    models/model_for_inference.pth

            # Restart Streamlit
            pkill -f streamlit || true
            nohup streamlit run app.py --server.port 8501 \
                  --server.address 0.0.0.0 > app.log 2>&1 &
          EOF
```

Results

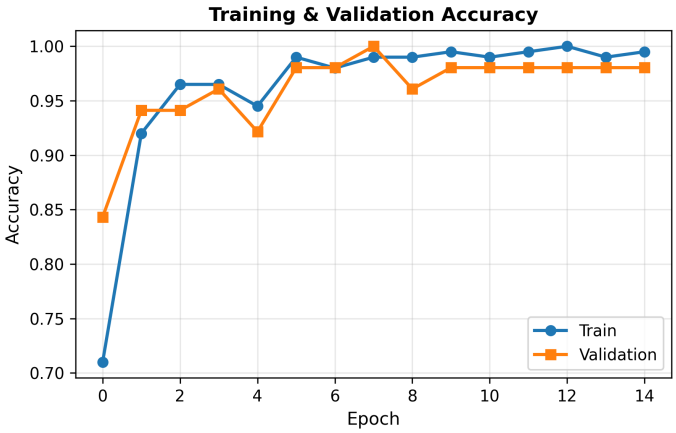
Final Model Performance (DenseNet-121 Fine-tuned)

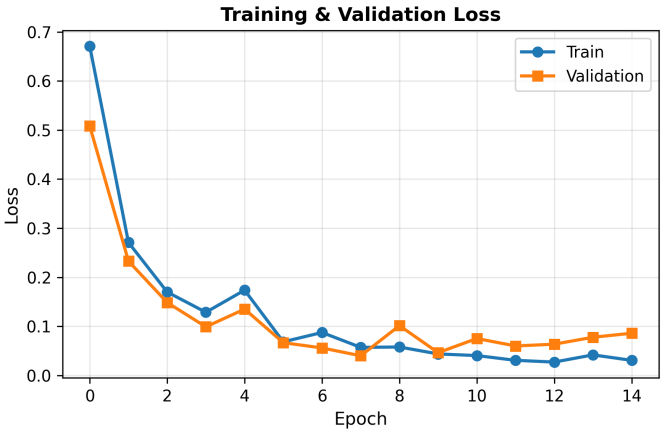
```
============================================================
TEST EVALUATION RESULTS
============================================================

Overall Metrics:
  Accuracy:  0.9394
  Precision: 0.9495 (weighted)
  Recall:    0.9394 (weighted)
  F1-Score:  0.9388 (weighted)

Per-Class Metrics:
------------------------------------------------------------
Covid                | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
Normal               | Precision: 1.0000 | Recall: 0.8000 | F1: 0.8889
Viral Pneumonia      | Precision: 0.8333 | Recall: 1.0000 | F1: 0.9091
```

Confusion Matrix

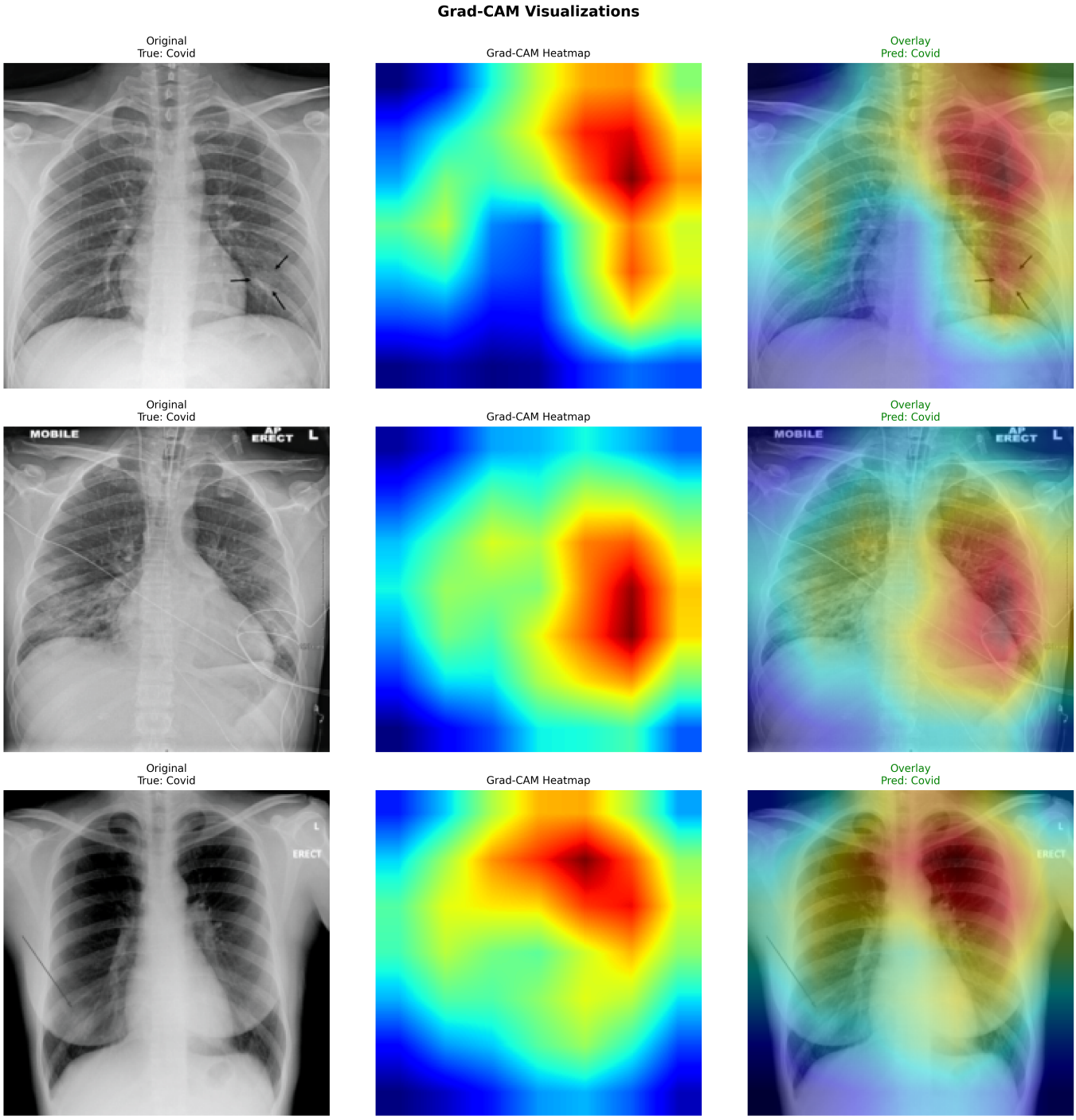


Model Training history





GradCAM Visualization



Model Comparison Summary

Best Model: DenseNet-121

- **Test Accuracy**: 93.94%
- **COVID-19 Detection**: 100% Precision & Recall
- **ROC-AUC**: 0.9917
- **Parameters**: ~7M (efficient)

Why DenseNet-121 Won:

1. Dense connections improve gradient flow
2. Feature reuse through concatenation

3. Excellent performance on medical images
4. Balanced accuracy across all classes

Setup Instructions

Local Setup

```
# Clone repository
git clone https://github.com/username/Multi-class-COVID-19-Detection.git
cd Multi-class-COVID-19-Detection

# Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Run Streamlit app
streamlit run app.py

# Or run FastAPI
uvicorn api:app --host 0.0.0.0 --port 8000
```

AWS EC2 Setup

```
# Connect to EC2
ssh -i key.pem ubuntu@ec2-ip-address

# Install Python and dependencies
sudo apt update
sudo apt install python3-pip python3-venv

# Clone and setup
git clone <repo-url>
cd Multi-class-COVID-19-Detection
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Configure AWS CLI
aws configure

# Download model from S3
aws s3 cp s3://bucket-name/models/model_for_inference.pth models/

# Run application
streamlit run app.py --server.port 8501 --server.address 0.0.0.0
```

Requirements.txt

```
torch==2.3.0+cpu
torchvision==0.18.0+cpu
-f https://download.pytorch.org/whl/cpu/torch_stable.html

streamlit
fastapi
uvicorn
boto3
Pillow
numpy
pandas
opencv-python-headless
matplotlib
seaborn
scikit-learn
```

Project Structure

```
Multi-class-COVID-19-Detection/
│
├── .github/workflows/
│   └── deploy.yml                    # CI/CD configuration
│
├── Covid19-dataset/
│   ├── train/                        # Training images
│   └── test/                         # Test images
│
├── models/
│   └── model_for_inference.pth       # Trained model
│
├── output/
│   ├── confusion_matrix.png
│   ├── roc_curves.png
│   └── training_history.png
│
├── Pretrained_model_comparison.ipynb   # Model comparison
├── finetuning_best_model.ipynb         # Fine-tuning
├── app.py                               # Streamlit app
├── api.py                               # FastAPI service
├── requirements.txt
└── README.md
```

Key Takeaways

Technical Achievements

☑ 93.94% test accuracy on COVID-19 classification ☑ Perfect precision/recall for COVID-19 class (1.0000) ☑ Grad-CAM explainability for clinical trust ☑ Production-ready deployment on AWS ☑ Automated CI/CD pipeline

Business Impact

- **Clinical Support**: Fast triage in hospitals
- **Remote Healthcare**: Diagnostic aid in low-resource areas
- **Scalability**: Cloud deployment for mass screening
- **Transparency**: Visual explanations for medical professionals

Skills Demonstrated

- Deep learning model development
- Transfer learning optimization
- Medical image analysis
- Cloud deployment (AWS)
- DevOps (CI/CD)
- API development
- Explainable AI
