Project Methodology: Aircraft X-Ray CNN Production Deployment

Executive Summary

This document outlines the methodology for deploying a ResNet50-based CNN for automated water ingression detection to production using PyTorch, FastAPI, and Hugging Face Spaces. This project represents the transition from research model development (Aircraft Flap X-Ray CNN Analysis) to production deployment, demonstrating complete model lifecycle management from training to live web application.

The deployment achieved 92.9% accuracy with confidence scoring and mobile-responsive interface, establishing a framework for AI-assisted aviation NDT operations.

1. Project Context & Technical Evolution

1.1 Deployment Positioning in Portfolio

Primary Objective: Demonstrate production deployment capabilities by transforming research CNN model into accessible web application with architecture and user experience.

Technical Progression:

• Original Implementation: TensorFlow/Keras research model (88.9% accuracy)

• Technology Migration: Complete transition to PyTorch for production consistency

• Web Application: FastAPI backend with JavaScript frontend

• Production Deployment: Hugging Face Spaces with Docker containerization

1.2 Business Case for Deployment

Operational Need: Bridge the gap between research model development and practical implementation for aviation NDT applications.

Technical Requirements:

• Accessible web interface for non-technical users

• Mobile-responsive design for field operations

• Confidence scoring for safety-critical decisions

• Error handling and performance management

2. Technology Migration Strategy

2.1 Framework Transition Rationale

TensorFlow to PyTorch Migration Drivers:

• Portfolio Consistency: All subsequent projects (Projects 5 & 6) implemented in PyTorch

• Deployment Simplicity: PyTorch state dict loading more straightforward than TensorFlow SavedModel

• Production Integration: Better compatibility with FastAPI serving frameworks

• Model Architecture Control: More granular control over custom layer implementations

2.2 Architecture Preservation

Model Equivalence Validation:

```
# Original TensorFlow approach
model = tf.keras.applications.ResNet50(...)
model.add(tf.keras.layers.Dense(...))


# PyTorch equivalent
class ResNetWaterDetector(nn.Module):
  def __init__(self, num_classes=1):
    self.backbone = models.resnet50(pretrained=True)
    self.classifier = nn.Sequential(...)
```

Performance Verification: 92.9% accuracy achieved with PyTorch implementation, representing 4% improvement over original TensorFlow model while maintaining identical architecture principles.

3. Web Application Architecture

3.1 Backend Implementation Strategy

FastAPI Selection Rationale:

- Performance: ASGI framework with async request handling
- Automatic Documentation: OpenAPI schema generation for API endpoints
- Type Safety: Pydantic models for request/response validation
- Ready: Built-in middleware support for CORS, authentication, monitoring

Model Serving Architecture:

```
# Global model loading for performance management
model = None
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
def load_model():
  global model
  if model is None:
    model = ResNetWaterDetector()
    model.load_state_dict(torch.load('model.pth', map_location=device))
    model.eval()
  return model
```

3.2 Frontend Development Strategy

Technology Choice: JavaScript implementation avoiding framework dependencies for deployment simplicity and performance management.

Key Features Implementation:

• Image Upload: Support for camera capture and gallery selection

• Interactive Cropping: Canvas-based ROI selection with drag-and-drop handles

• Mobile Management: Touch event handling and responsive design

• Real-time Preview: Immediate visual feedback for user interactions

4. Production Deployment Framework

4.1 Hugging Face Spaces Selection

Platform Advantages:

• Containerization: Docker-based deployment with automatic scaling

• GPU Acceleration: CUDA support for model inference management

• Version Control: Git-based deployment with automatic updates

• Public Access: Shareable URLs for portfolio demonstration

• Minimal Configuration: Setup requirements for immediate deployment

4.2 Containerization Strategy

Deployment Architecture:

```
# app.py - FastAPI application entry point

app = FastAPI()

app.mount("/static", StaticFiles(directory="."), name="static")


@app.get("/", response_class=HTMLResponse)

async def read_root():

  return HTMLResponse(content=html_content)


@app.post("/predict")

async def predict(image: UploadFile = File(...)):

  # Model inference pipeline
```

Resource Management: Single-file model deployment with efficient memory management and error handling for production stability.

5. User Experience Design

5.1 Safety-Critical Interface Design

Decision Framework:

• High Confidence (≥80%): Automated decision capability

• Medium Confidence (60-80%): Confirmation needed

• Low Confidence (<60%): Manual review recommended

Visual Feedback Strategy:

• Color-coded confidence levels for immediate interpretation

• Clear prediction text with percentage confidence display

• Contextual guidance for operational decision-making

5.2 Mobile-First Development

Responsive Design Implementation:

• Touch-optimized controls for field operations

• Camera API integration for direct image capture

• Adaptive layout for various screen sizes

• Enhancement from mobile to desktop

Cross-platform Support:

• iOS Safari support with proper touch event handling

• Android Chrome management for camera access

• Desktop browser fallback with enhanced features

6. Model Integration & Performance

6.1 Preprocessing Pipeline Consistency

Training-Inference Alignment:

```
def preprocess_image(image):
  transform = transforms.Compose([
    transforms.Resize((IMG_HEIGHT, IMG_WIDTH)),
    transforms.ToTensor(),
    transforms.Lambda(lambda x: x.repeat(3, 1, 1)),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
  ])
  return transform(image.convert('L')).unsqueeze(0)
```

Critical Requirement: Identical preprocessing between training and inference to maintain model performance and avoid prediction drift.

6.2 Performance Management

Inference Efficiency:

• Model loading: Single initialization at application startup

• GPU utilization: Automatic device detection and allocation

• Memory management: Efficient tensor operations with proper cleanup

• Error handling: Graceful degradation for various image formats and sizes

7. Security & Error Handling

7.1 Error Management

Exception Handling:

```
try:
    prediction = model(processed_image).item()
    predicted_class = 'Water Detected' if prediction > 0.5 else 'No Water Ingression'
    return JSONResponse({...})
except Exception as e:
    raise HTTPException(status_code=500, detail=str(e))
```

Robust File Processing: Support for multiple image formats with validation and automatic conversion to ensure consistent processing pipeline.

7.2 Input Validation Framework

Request Validation:

- File type verification for image uploads
- Size limitations to prevent resource exhaustion
- Format conversion for preprocessing compatibility
- Error messaging for user guidance

8. Confidence Scoring Implementation

8.1 Aviation Safety Framework

Approach: Three-tier confidence classification designed for safety-critical applications where false negatives carry higher operational risk than false positives.

Operational Integration:

- Clear confidence thresholds for decision support
- Visual indicators for immediate interpretation
- Contextual recommendations for operational procedures

8.2 Prediction Interpretation

Implementation:

```
def interpret_prediction(confidence_score):
    if confidence_score > 0.8:
        return f"HIGH CONFIDENCE: {confidence_score:.1%}"
    elif confidence_score > 0.6:
        return f"MEDIUM CONFIDENCE: {confidence_score:.1%}"
    else:
        return f"LOW CONFIDENCE: {confidence_score:.1%} - Manual review recommended"
```

9. Deployment Validation & Testing

9.1 Readiness Assessment

Performance Validation: Model inference speed, memory utilization, and response time management for production workloads.

Functionality Testing:

• Cross-browser compatibility verification

• Mobile device testing across iOS and Android platforms

• Image format support validation

• Error condition handling verification

9.2 User Acceptance Criteria

Interface Validation:

• Intuitive image upload and cropping workflow

• Clear prediction display with confidence scoring

• Responsive design across device types

• Accessibility considerations for operational use

10. Business Value & Operational Impact

10.1 Demonstration Value Portfolio Enhancement

Complete model lifecycle demonstration from research development through production deployment, showcasing full-stack data science capabilities.

Technical Competency: Integration of machine learning, web development, and production deployment skills in unified application.

10.2 Future Enhancement Pathway Scalability Considerations

• Database integration for prediction logging and analytics

• User authentication for multi-tenant deployment

• API versioning for backward compatibility

• Performance monitoring and alerting systems

Integration Readiness: Architecture designed for future integration with aviation maintenance management systems and digital radiography workflows.

11. Lessons Learned & Best Practices

11.1 Technical Insights

Framework Migration: PyTorch transition provided cleaner deployment pipeline compared to TensorFlow SavedModel approach, confirming strategic technology choice for production applications.

Preprocessing Consistency: Critical importance of exact preprocessing pipeline alignment between training and inference to maintain model performance.

11.2 Deployment Considerations

Platform Selection: Hugging Face Spaces provided optimal balance of functionality, performance, and accessibility for portfolio demonstration purposes.

User Experience: Mobile-first design essential for field applications, with enhancement for desktop environments.

Conclusion

This methodology demonstrates successful transition from research model development to production deployment, encompassing technology migration, web application development, and production hosting. The systematic approach ensures model performance preservation while creating accessible user interface for practical aviation NDT applications.

Key Methodological Achievements:

- Complete technology stack migration maintaining model performance
- Web application with mobile-responsive design
- Confidence scoring framework for safety-critical applications
- Architecture designed for system integration
- Error handling and performance management

The project establishes a framework for ML model deployment in specialized industrial applications while demonstrating development capabilities essential for data science roles requiring production implementation skills.