Smart Document Intelligence System - Complete Project Plan

📋 Project Overview

Project Name: Smart Document Intelligence System

Goal: Build a production-ready ML system that classifies documents and extracts key information

Timeline: 5 Sprints (5 weeks)

Deployment: AWS SageMaker + Web Application

Tech Stack: Python, PyTorch, SageMaker, Streamlit, Docker

🎯 What We're Building

Document Types (4 Classes)

1. **Invoices** - Extract: amount, vendor, date, invoice number
2. **Receipts** - Extract: merchant, total, date, items
3. **Resumes** - Extract: name, skills, experience, education
4. **Contracts** - Extract: parties, dates, key terms

Core Features

- Upload document (PDF/Image)
- Classify document type (98%+ accuracy)
- Extract key information automatically
- Display structured results
- Export results (JSON/CSV)

📊 Data Sources & Models

Datasets (All FREE & Public)

1. **RVL-CDIP Dataset** - 400,000 document images, 16 classes
   - Source: https://huggingface.co/datasets/aharley/rvl_cdip
   - Size: ~100GB (we'll use subset)
   - Classes: invoice, resume, email, memo, letter, etc.
2. **SROIE Dataset** - 1,000 receipt images with annotations
   - Source: https://rrc.cvc.uab.es/?ch=13

    - Size: ~500MB
    - Perfect for receipt extraction

3. **Resume Dataset** - Kaggle
    - Source: https://www.kaggle.com/datasets (search "resume")
    - Alternative: Generate synthetic resumes

4. **Contract Samples** - Public contracts
    - Source: EDGAR SEC filings (public contracts)
    - Alternative: Use RVL-CDIP contract samples

Model Architecture

Primary Model: LayoutLMv3 (Microsoft)

- **Why**: State-of-the-art for document understanding (95%+ accuracy)
- **Advantages**: Understands text + layout + visual features
- **Pre-trained**: Yes (on millions of documents)
- **License**: Check HuggingFace (some have commercial restrictions)
- **Alternative**: Donut (MIT license, fully open)

Backup/Comparison Models:

- ResNet50 (for pure image classification)
- BERT (for text-only extraction)

🏗️ Project Structure

```
document-intelligence-ml/
├── README.md
├── requirements.txt
├── .env.example
├── .gitignore
├── setup.py
│
├── data/
│   ├── raw/            # Downloaded datasets
│   ├── processed/      # Cleaned data
│   └── samples/        # Test samples
│
├── notebooks/
```

```
│   ├─── sprint1_eda.ipynb
│   ├─── sprint2_preprocessing.ipynb
│   ├─── sprint3_training.ipynb
│   ├─── sprint4_evaluation.ipynb
│   └─── sprint5_deployment.ipynb
│
├─── src/
│   ├─── __init__.py
│   ├─── config.py
│   │
│   ├─── data/
│   │   ├─── __init__.py
│   │   ├─── download.py
│   │   ├─── preprocess.py
│   │   └─── augmentation.py
│   │
│   ├─── models/
│   │   ├─── __init__.py
│   │   ├─── classifier.py
│   │   ├─── extractor.py
│   │   └─── inference.py
│   │
│   ├─── training/
│   │   ├─── __init__.py
│   │   ├─── train.py
│   │   ├─── evaluate.py
│   │   └─── metrics.py
│   │
│   ├─── pipeline/
│   │   ├─── __init__.py
│   │   ├─── sagemaker_pipeline.py
│   │   └─── step_functions.py
│   │
│   └─── deployment/
│      ├─── __init__.py
│      ├─── lambda_functions/
│      ├─── api.py
│      └─── monitoring.py
│
├─── web_app/
│   ├─── app.py
│   ├─── requirements.txt
│   ├─── Dockerfile
│   ├─── static/
│   │   ├─── css/
│   │   └─── js/
│   └─── templates/
```

```
│       ├── index.html
│       └── results.html
│
├── tests/
│   ├── test_data.py
│   ├── test_models.py
│   └── test_api.py
│
├── deployment/
│   ├── cloudformation/
│   ├── terraform/
│   └── docker-compose.yml
│
└── docs/
    ├── api_documentation.md
    ├── model_card.md
    └── blog_post.md
```

🚀 SPRINT BREAKDOWN

SPRINT 0: SETUP & PREPARATION (Before Sprint 1)

Tasks

1. Environment Setup (1 hour)

- [ ] Install Python 3.9+
- [ ] Install AWS CLI
- [ ] Configure AWS credentials
- [ ] Install required packages

Commands:

```bash
```

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

# Install packages
pip install --upgrade pip
pip install boto3 sagemaker pandas numpy matplotlib seaborn
pip install torch torchvision transformers datasets
pip install scikit-learn opencv-python pillow pytesseract
pip install streamlit plotly jupyterlab
```

2. AWS Setup (1 hour)

- [ ] Create AWS account (if needed)
- [ ] Create S3 bucket: `document-intelligence-data`
- [ ] Create IAM role for SageMaker
- [ ] Set up SageMaker Studio or Notebook instance

3. GitHub Repository Setup (30 min)

- [ ] Create repository: `document-intelligence-ml`
- [ ] Initialize with README
- [ ] Create project structure
- [ ] Set up .gitignore

Initial Files to Create:

requirements.txt:

```
boto3==1.28.0
sagemaker==2.180.0
torch==2.0.1
transformers==4.35.0
datasets==2.14.0
pandas==2.1.0
numpy==1.24.0
matplotlib==3.7.0
seaborn==0.12.0
scikit-learn==1.3.0
opencv-python==4.8.0
Pillow==10.0.0
streamlit==1.28.0
pytesseract==0.3.10
python-dotenv==1.0.0
```

config.py:

```python
import os
from pathlib import Path

# Project paths
PROJECT_ROOT = Path(__file__).parent.parent
DATA_DIR = PROJECT_ROOT / "data"
RAW_DATA_DIR = DATA_DIR / "raw"
PROCESSED_DATA_DIR = DATA_DIR / "processed"
MODEL_DIR = PROJECT_ROOT / "models"

# AWS Configuration
AWS_REGION = os.getenv("AWS_REGION", "us-east-1")
S3_BUCKET = os.getenv("S3_BUCKET", "document-intelligence-data")
SAGEMAKER_ROLE = os.getenv("SAGEMAKER_ROLE")

# Model Configuration
DOCUMENT_CLASSES = ["invoice", "receipt", "resume", "contract"]
IMAGE_SIZE = (224, 224)
BATCH_SIZE = 16
LEARNING_RATE = 2e-5
EPOCHS = 5

# Extraction Fields
EXTRACTION_CONFIG = {
    "invoice": ["invoice_number", "date", "vendor", "total_amount"],
    "receipt": ["merchant", "date", "total", "items"],
    "resume": ["name", "email", "phone", "skills", "experience"],
    "contract": ["parties", "effective_date", "expiration_date", "terms"]
}
```

SPRINT 1: DATA COLLECTION & EXPLORATION (Week 1)

Goal: Understand the data, download datasets, perform EDA

Duration: 5 days

Output: Clean dataset ready for training

Day 1: Dataset Research & Download Strategy

Task 1.1: Research Datasets (2 hours)

- [ ] Review RVL-CDIP dataset documentation

- [ ] Review SROIE dataset
- [ ] Identify resume dataset on Kaggle
- [ ] Document dataset statistics

Deliverable: `docs/dataset_research.md`

Task 1.2: Create Download Scripts (3 hours)

- [ ] Write script to download RVL-CDIP subset
- [ ] Write script to download SROIE
- [ ] Write script to download resume dataset
- [ ] Add progress tracking

Code: `src/data/download.py`

python

```python
from datasets import load_dataset
import boto3
from tqdm import tqdm

def download_rvl_cdip(num_samples_per_class=1000):
    """Download subset of RVL-CDIP dataset"""
    print("Downloading RVL-CDIP dataset...")

    # Load from HuggingFace
    dataset = load_dataset("aharley/rvl_cdip", split="train")

    # Filter for our 4 classes
    target_classes = ['invoice', 'resume', 'letter', 'email']

    # Sample and save
    # ... implementation

    return dataset

def download_sroie():
    """Download SROIE receipt dataset"""
    # Implementation
    pass

def upload_to_s3(local_path, s3_path):
    """Upload data to S3"""
    s3 = boto3.client('s3')
    # Implementation
    pass
```

Task 1.3: Download Data (2 hours)

- [ ] Run download scripts
- [ ] Verify data integrity
- [ ] Upload to S3
- [ ] Document dataset structure

Success Criteria:

- At least 4,000 images downloaded (1,000 per class)
- Data uploaded to S3
- No corrupted files

Day 2-3: Exploratory Data Analysis

Task 1.4: Create EDA Notebook (4 hours)

- [ ] Load datasets
- [ ] Visualize sample images
- [ ] Analyze class distribution
- [ ] Check image dimensions
- [ ] Analyze file formats
- [ ] Identify data quality issues

Notebook: `notebooks/sprint1_eda.ipynb`

Key Analyses:

```python
```

```python
# 1. Class Distribution
import matplotlib.pyplot as plt
import seaborn as sns

class_counts = df['label'].value_counts()
plt.figure(figsize=(10, 6))
sns.barplot(x=class_counts.index, y=class_counts.values)
plt.title('Document Class Distribution')
plt.show()

# 2. Image Size Distribution
widths = []
heights = []
for img_path in tqdm(image_paths):
    img = Image.open(img_path)
    widths.append(img.width)
    heights.append(img.height)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.hist(widths, bins=50)
plt.title('Image Widths')
plt.subplot(1, 2, 2)
plt.hist(heights, bins=50)
plt.title('Image Heights')
plt.show()

# 3. Sample Visualizations
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
for i, doc_class in enumerate(DOCUMENT_CLASSES):
    # Show 2 examples per class
    pass
```

Task 1.5: Data Quality Report (2 hours)

- [ ] Document findings
- [ ] Identify preprocessing needs
- [ ] Plan data cleaning strategy

Deliverable: `docs/data_quality_report.md`

Day 4-5: Data Preprocessing

Task 1.6: Build Preprocessing Pipeline (6 hours)

- [ ] Create image preprocessing functions
- [ ] Handle different image formats
- [ ] Resize images
- [ ] Normalize images
- [ ] Create train/val/test splits

Code: `src/data/preprocess.py`

```python
```

```python
import cv2
from PIL import Image
import numpy as np

class DocumentPreprocessor:
    def __init__(self, target_size=(224, 224)):
        self.target_size = target_size

    def preprocess_image(self, image_path):
        """Preprocess single image"""
        # Load image
        img = Image.open(image_path).convert('RGB')

        # Resize
        img = img.resize(self.target_size)

        # Convert to array
        img_array = np.array(img)

        # Normalize
        img_array = img_array / 255.0

        return img_array

    def create_splits(self, df, train_ratio=0.7, val_ratio=0.15):
        """Create train/val/test splits"""
        # Stratified split
        from sklearn.model_selection import train_test_split

        train_df, temp_df = train_test_split(
            df, train_size=train_ratio, stratify=df['label']
        )
        val_df, test_df = train_test_split(
            temp_df, train_size=val_ratio/(1-train_ratio),
            stratify=temp_df['label']
        )

        return train_df, val_df, test_df
```

Task 1.7: Process All Data (4 hours)

- [ ] Run preprocessing on all images
- [ ] Save processed data
- [ ] Create data manifest files
- [ ] Upload to S3

Task 1.8: Create Data Loaders (2 hours)

- [ ] Build PyTorch Dataset class
- [ ] Build DataLoader
- [ ] Test loading speed
- [ ] Add data augmentation

Code Example:

```python
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms

class DocumentDataset(Dataset):
    def __init__(self, df, transform=None):
        self.df = df
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        img_path = self.df.iloc[idx]['image_path']
        label = self.df.iloc[idx]['label']

        # Load image
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        return image, label

# Augmentation
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation(5),
    transforms.ColorJitter(0.2, 0.2, 0.2),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

Sprint 1 Deliverables Checklist

- [ ] Dataset downloaded (4,000+ images)
- [ ] EDA notebook completed
- [ ] Data quality report
- [ ] Preprocessing pipeline built
- [ ] Train/val/test splits created
- [ ] Data uploaded to S3
- [ ] DataLoaders tested and working

Success Metrics:

- 1,000 images per class minimum
- Clean 70/15/15 split
- No corrupted images
- Average preprocessing time < 100ms per image

SPRINT 2: MODEL DEVELOPMENT (Week 2)

Goal: Build classification model, achieve 90%+ accuracy
Duration: 5 days
Output: Trained document classifier

Day 1: Model Architecture Setup

Task 2.1: Research Model Options (2 hours)

- [ ] Review LayoutLMv3 documentation
- [ ] Check model licenses
- [ ] Compare alternatives (Donut, ResNet)
- [ ] Make architecture decision

Task 2.2: Setup Base Model (4 hours)

- [ ] Load pre-trained LayoutLMv3
- [ ] Modify for 4-class classification
- [ ] Test forward pass
- [ ] Calculate model size

Code: `src/models/classifier.py`

```python
```

```python
import torch
import torch.nn as nn
from transformers import LayoutLMv3ForSequenceClassification, LayoutLMv3Processor

class DocumentClassifier(nn.Module):
    def __init__(self, num_classes=4, model_name="microsoft/layoutlmv3-base"):
        super().__init__()

        # Load pre-trained model
        self.model = LayoutLMv3ForSequenceClassification.from_pretrained(
            model_name,
            num_labels=num_classes
        )

        self.processor = LayoutLMv3Processor.from_pretrained(model_name)

    def forward(self, images, words=None, boxes=None):
        # Process inputs
        encoding = self.processor(
            images=images,
            text=words,
            boxes=boxes,
            return_tensors="pt",
            padding="max_length",
            truncation=True
        )

        # Forward pass
        outputs = self.model(**encoding)
        return outputs

# Alternative: Simple CNN if LayoutLM too heavy
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=4):
        super().__init__()
        # Use ResNet backbone
        import torchvision.models as models
        self.backbone = models.resnet50(pretrained=True)

        # Replace final layer
        num_features = self.backbone.fc.in_features
        self.backbone.fc = nn.Linear(num_features, num_classes)

    def forward(self, x):
        return self.backbone(x)
```

Day 2-3: Training Pipeline

Task 2.3: Build Training Script (6 hours)

- [ ] Create training loop
- [ ] Add validation loop
- [ ] Implement early stopping
- [ ] Add checkpointing
- [ ] Add logging

Code: `src/training/train.py`

```python
```

```python
import torch
from torch.optim import AdamW
from tqdm import tqdm

class Trainer:
    def __init__(self, model, train_loader, val_loader, device='cuda'):
        self.model = model.to(device)
        self.train_loader = train_loader
        self.val_loader = val_loader
        self.device = device

        # Optimizer
        self.optimizer = AdamW(model.parameters(), lr=2e-5)

        # Loss
        self.criterion = nn.CrossEntropyLoss()

        # Best model tracking
        self.best_val_acc = 0

    def train_epoch(self):
        self.model.train()
        total_loss = 0
        correct = 0
        total = 0

        for images, labels in tqdm(self.train_loader):
            images = images.to(self.device)
            labels = labels.to(self.device)

            # Forward
            outputs = self.model(images)
            loss = self.criterion(outputs.logits, labels)

            # Backward
            self.optimizer.zero_grad()
            loss.backward()
            self.optimizer.step()

            # Metrics
            total_loss += loss.item()
            _, predicted = outputs.logits.max(1)
            correct += predicted.eq(labels).sum().item()
            total += labels.size(0)

        return total_loss / len(self.train_loader), correct / total
```

```python
    def validate(self):
        self.model.eval()
        correct = 0
        total = 0

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

                outputs = self.model(images)
                _, predicted = outputs.logits.max(1)
                correct += predicted.eq(labels).sum().item()
                total += labels.size(0)

        return correct / total

    def fit(self, epochs=10):
        for epoch in range(epochs):
            print(f"Epoch {epoch+1}/{epochs}")

            # Train
            train_loss, train_acc = self.train_epoch()
            print(f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}")

            # Validate
            val_acc = self.validate()
            print(f"Val Acc: {val_acc:.4f}")

            # Save best model
            if val_acc > self.best_val_acc:
                self.best_val_acc = val_acc
                torch.save(self.model.state_dict(), 'best_model.pth')
                print("✓ Saved best model")
```

Task 2.4: Train Initial Model (4 hours)

- [ ] Start training with baseline config
- [ ] Monitor training metrics
- [ ] Save training logs
- [ ] Evaluate on validation set

Day 4: Model Evaluation

Task 2.5: Build Evaluation Script (3 hours)

- [ ] Calculate accuracy, precision, recall, F1
- [ ] Generate confusion matrix
- [ ] Analyze per-class performance
- [ ] Identify failure cases

Code: `src/training/evaluate.py`

```python
```

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

def evaluate_model(model, test_loader, device='cuda'):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            outputs = model(images)
            _, predicted = outputs.logits.max(1)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.numpy())

    # Classification Report
    print(classification_report(all_labels, all_preds,
                                target_names=DOCUMENT_CLASSES))

    # Confusion Matrix
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=DOCUMENT_CLASSES,
                yticklabels=DOCUMENT_CLASSES)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.title('Confusion Matrix')
    plt.savefig('confusion_matrix.png')

    return all_preds, all_labels
```

Task 2.6: Run Full Evaluation (2 hours)

- [ ] Test on test set
- [ ] Generate metrics
- [ ] Create visualizations
- [ ] Document results

Day 5: Model Optimization

Task 2.7: Hyperparameter Tuning (4 hours)

- [ ] Try different learning rates
- [ ] Experiment with batch sizes
- [ ] Test different architectures
- [ ] Use SageMaker automatic tuning (optional)

Task 2.8: Model Selection (2 hours)

- [ ] Compare all model versions
- [ ] Select best performing model
- [ ] Save final model
- [ ] Upload to S3 / SageMaker Model Registry

Sprint 2 Deliverables Checklist

- [ ] Document classifier trained
- [ ] Validation accuracy > 90%
- [ ] Test accuracy > 88%
- [ ] Confusion matrix generated
- [ ] Model saved to S3
- [ ] Training notebook completed
- [ ] Evaluation report written

Success Metrics:

- Overall accuracy: > 90%
- Per-class F1 score: > 0.85
- Inference time: < 500ms per document
- Model size: < 500MB

SPRINT 3: INFORMATION EXTRACTION (Week 3)

Goal: Build extraction models for each document type

Duration: 5 days

Output: Working extraction pipeline

Day 1-2: OCR & Text Extraction

Task 3.1: Setup OCR Engine (3 hours)

- [ ] Install Tesseract
- [ ] Test AWS Textract
- [ ] Compare OCR engines
- [ ] Choose best option

Code: `src/models/ocr.py`

```python
import pytesseract
from PIL import Image
import boto3

class OCREngine:
    def __init__(self, engine='tesseract'):
        self.engine = engine
        if engine == 'textract':
            self.textract = boto3.client('textract')

    def extract_text(self, image_path):
        if self.engine == 'tesseract':
            img = Image.open(image_path)
            text = pytesseract.image_to_string(img)
            boxes = pytesseract.image_to_boxes(img)
            return text, boxes

        elif self.engine == 'textract':
            with open(image_path, 'rb') as document:
                response = self.textract.detect_document_text(
                    Document={'Bytes': document.read()}
                )
            return self._parse_textract_response(response)
```

Task 3.2: Build Entity Extraction (5 hours)

- [ ] Setup spaCy or BERT NER
- [ ] Define entities for each document type
- [ ] Create extraction rules
- [ ] Test extraction accuracy

Code: `src/models/extractor.py`

python

```python
import spacy
import re
from datetime import datetime

class InformationExtractor:
    def __init__(self):
        # Load NER model
        self.nlp = spacy.load("en_core_web_sm")

    def extract_invoice(self, text):
        """Extract invoice information"""
        results = {}

        # Invoice number (pattern: INV-XXXXX)
        inv_pattern = r'INV-?\d{4,}'
        inv_match = re.search(inv_pattern, text, re.IGNORECASE)
        if inv_match:
            results['invoice_number'] = inv_match.group()

        # Total amount
        amount_pattern = r'\$\s*[\d,]+\.?\d*'
        amounts = re.findall(amount_pattern, text)
        if amounts:
            results['total_amount'] = amounts[-1]  # Last amount usually total

        # Date
        date_pattern = r'\d{1,2}[/-]\d{1,2}[/-]\d{2,4}'
        date_match = re.search(date_pattern, text)
        if date_match:
            results['date'] = date_match.group()

        # Vendor (using NER)
        doc = self.nlp(text)
        for ent in doc.ents:
            if ent.label_ == 'ORG':
                results['vendor'] = ent.text
                break

        return results

    def extract_receipt(self, text):
        """Extract receipt information"""
        # Similar logic for receipts
        pass

    def extract_resume(self, text):
```

```
    """Extract resume information"""
    results = {}

    # Email
    email_pattern = r'[\w\.-]+@[\w\.-]+\.\w+'
    email_match = re.search(email_pattern, text)
    if email_match:
        results['email'] = email_match.group()

    # Phone
    phone_pattern = r'\+?1?\s*\(?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}'
    phone_match = re.search(phone_pattern, text)
    if phone_match:
        results['phone'] = phone_match.group()

    # Skills (simple keyword matching)
    skill_keywords = ['python', 'java', 'javascript', 'sql', 'aws',
                     'machine learning', 'data science']
    found_skills = []
    text_lower = text.lower()
    for skill in skill_keywords:
        if skill in text_lower:
            found_skills.append(skill)
    results['skills'] = found_skills

    return results

def extract_contract(self, text):
    """Extract contract information"""
    # Similar logic for contracts
    pass
```

Day 3-4: Extraction Pipeline Integration

Task 3.3: Build End-to-End Pipeline (6 hours)

- [ ] Combine classifier + OCR + extractor
- [ ] Handle different document types
- [ ] Add error handling
- [ ] Test pipeline

Code: `src/pipeline/document_pipeline.py`

```
python
```

```python
class DocumentPipeline:
    def __init__(self):
        self.classifier = DocumentClassifier.load('best_model.pth')
        self.ocr = OCREngine(engine='tesseract')
        self.extractor = InformationExtractor()

    def process_document(self, image_path):
        """Process single document end-to-end"""
        # Step 1: Classify
        doc_type, confidence = self.classifier.predict(image_path)

        # Step 2: OCR
        text, boxes = self.ocr.extract_text(image_path)

        # Step 3: Extract based on type
        if doc_type == 'invoice':
            extracted_info = self.extractor.extract_invoice(text)
        elif doc_type == 'receipt':
            extracted_info = self.extractor.extract_receipt(text)
        elif doc_type == 'resume':
            extracted_info = self.extractor.extract_resume(text)
        elif doc_type == 'contract':
            extracted_info = self.extractor.extract_contract(text)

        # Return results
        return {
            'document_type': doc_type,
            'confidence': confidence,
            'extracted_info': extracted_info,
            'full_text': text
        }
```

Task 3.4: Test on Real Documents (4 hours)

- [ ] Test on 100 documents
- [ ] Calculate extraction accuracy
- [ ] Identify common errors
- [ ] Improve extraction rules

Day 5: Validation & Optimization

Task 3.5: Validate Extraction Quality (4 hours)

- [ ] Manual review of extractions

- [ ] Calculate precision/recall per field
- [ ] Fix critical bugs
- [ ] Document limitations

Task 3.6: Optimize Performance (2 hours)

- [ ] Profile code
- [ ] Optimize slow functions
- [ ] Add caching where appropriate

Sprint 3 Deliverables Checklist

- [ ] OCR working on all document types
- [ ] Extraction models for all 4 types
- [ ] End-to-end pipeline functional
- [ ] Extraction accuracy > 80% per field
- [ ] Pipeline tested on 100+ documents
- [ ] Performance optimizations complete

Success Metrics:

- OCR accuracy: > 95%
- Field extraction accuracy: > 80%
- Pipeline processing time: < 2 seconds per document
- Error handling robust

SPRINT 4: SAGEMAKER DEPLOYMENT (Week 4)

Goal: Deploy models to AWS SageMaker
Duration: 5 days
Output: Production-ready ML endpoints

Day 1: SageMaker Pipeline Setup

Task 4.1: Create Training Job (4 hours)

- [ ] Convert training script to SageMaker format
- [ ] Configure instance types
- [ ] Set hyperparameters
- [ ] Test training job

Code: `src/pipeline/sagemaker_pipeline.py`

```python
import sagemaker
from sagemaker.pytorch import PyTorch

def create_training_job():
    sagemaker_session = sagemaker.Session()
    role = sagemaker.get_execution_role()

    # Create PyTorch estimator
    estimator = PyTorch(
        entry_point='train.py',
        source_dir='src/training',
        role=role,
        framework_version='2.0.0',
        py_version='py310',
        instance_count=1,
        instance_type='ml.p3.2xlarge',  # GPU instance
        hyperparameters={
            'epochs': 10,
            'batch-size': 16,
            'learning-rate': 2e-5
        }
    )

    # Start training
    estimator.fit({'training': 's3://bucket/data/train'})

    return estimator
```

Task 4.2: Setup Model Registry (2 hours)

- [ ] Register trained models
- [ ] Add model metadata
- [ ] Version models
- [ ] Test model retrieval

Day 2: Model Deployment

Task 4.3: Create Inference Script (3 hours)

- [ ] Write inference.py
- [ ] Handle preprocessing
- [ ] Handle postprocessing

- [ ] Test locally

Code: `src/deployment/inference.py`

```python
import torch
import json

def model_fn(model_dir):
    """Load model"""
    model = DocumentClassifier()
    model.load_state_dict(torch.load(f"{model_dir}/model.pth"))
    return model

def input_fn(request_body, content_type):
    """Preprocess input"""
    if content_type == 'application/json':
        data = json.loads(request_body)
        # Process image
        return data
    else:
        raise ValueError(f"Unsupported content type: {content_type}")

def predict_fn(input_data, model):
    """Make prediction"""
    model.eval()
    with torch.no_grad():
        output = model(input_data)
    return output

def output_fn(prediction, accept):
    """Format output"""
    if accept == 'application/json':
        return json.dumps(prediction), accept
    raise ValueError(f"Unsupported accept type: {accept}")
```

Task 4.4: Deploy Model Endpoint (3 hours)

- [ ] Create endpoint configuration
- [ ] Deploy endpoint
- [ ] Test endpoint
- [ ] Monitor deployment

Code:

```python
```

```python
from sagemaker.pytorch import PyTorchModel

def deploy_model(estimator):
    pytorch_model = PyTorchModel(
        model_data=estimator.model_data,
        role=role,
        entry_point='inference.py',
        framework_version='2.0.0',
        py_version='py310'
    )

    predictor = pytorch_model.deploy(
        instance_type='ml.m5.large',
        initial_instance_count=1,
        endpoint_name='doc-classifier-endpoint'
    )

    return predictor
```

Day 3-4: Lambda Functions & API

Task 4.5: Create Lambda Functions (6 hours)

- [ ] Upload handler
- [ ] Classification handler
- [ ] Extraction handler
- [ ] Result formatter

Code: `src/deployment/lambda_functions/handler.py`

```python
python
```

```python
import boto3
import json

s3 = boto3.client('s3')
sagemaker_runtime = boto3.client('sagemaker-runtime')

def lambda_handler(event, context):
    # Get uploaded file
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Download file
    obj = s3.get_object(Bucket=bucket, Key=key)
    image_bytes = obj['Body'].read()

    # Call SageMaker endpoint
    response = sagemaker_runtime.invoke_endpoint(
        EndpointName='doc-classifier-endpoint',
        ContentType='application/x-image',
        Body=image_bytes
    )

    # Parse results
    result = json.loads(response['Body'].read())

    # Store results in DynamoDB
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('DocumentResults')
    table.put_item(Item={
        'document_id': key,
        'result': result
    })

    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }
```

Task 4.6: Setup API Gateway (3 hours)

- [ ] Create REST API
- [ ] Configure endpoints
- [ ] Add authentication
- [ ] Test API

Day 5: Monitoring & Testing

Task 4.7: Setup Model Monitoring (3 hours)

- [ ] Enable SageMaker Model Monitor
- [ ] Set up CloudWatch dashboards
- [ ] Configure alerts
- [ ] Test monitoring

Task 4.8: Load Testing (3 hours)

- [ ] Run load tests
- [ ] Measure latency
- [ ] Test auto-scaling
- [ ] Document performance

Sprint 4 Deliverables Checklist

- [ ] Models deployed to SageMaker
- [ ] Endpoints responding correctly
- [ ] Lambda functions working
- [ ] API Gateway configured
- [ ] Monitoring active
- [ ] Load testing complete

Success Metrics:

- Endpoint latency: < 1 second
- API success rate: > 99%
- Auto-scaling working
- Monitoring dashboards live

SPRINT 5: WEB APPLICATION & DOCUMENTATION (Week 5)

Goal: Build user-facing app and complete documentation

Duration: 5 days

Output: Deployed web app + blog post

Day 1-2: Web Application

Task 5.1: Build Streamlit App (8 hours)

- [ ] Create UI layout
- [ ] Add file upload
- [ ] Connect to API
- [ ] Display results
- [ ] Add export functionality

Code: `web_app/app.py`

```python
```

```python
import streamlit as st
import requests
from PIL import Image
import json

st.set_page_config(page_title="Document Intelligence", layout="wide")

st.title("📄 Smart Document Intelligence System")
st.markdown("Upload a document to automatically classify and extract key information")

# Sidebar
st.sidebar.header("About")
st.sidebar.info("""
This system can process:
- Invoices
- Receipts
- Resumes
- Contracts
""")

# File upload
uploaded_file = st.file_uploader(
    "Upload Document (PDF, PNG, JPG)",
    type=['pdf', 'png', 'jpg', 'jpeg']
)

if uploaded_file:
    # Display image
    col1, col2 = st.columns(2)

    with col1:
        st.subheader("Uploaded Document")
        image = Image.open(uploaded_file)
        st.image(image, use_column_width=True)

    # Process button
    if st.button("Analyze Document", type="primary"):
        with st.spinner("Processing..."):
            # Call API
            files = {'file': uploaded_file.getvalue()}
            response = requests.post(
                'https://api.yourdomain.com/analyze',
                files=files
            )

            results = response.json()
```

```python
    # Display results
    with col2:
        st.subheader("Analysis Results")

        # Document type
        doc_type = results['document_type']
        confidence = results['confidence']

        st.metric("Document Type", doc_type.capitalize())
        st.metric("Confidence", f"{confidence*100:.1f}%")

        # Extracted information
        st.subheader("Extracted Information")
        extracted = results['extracted_info']

        for key, value in extracted.items():
            st.text_input(
                key.replace('_', ' ').title(),
                value,
                key=key
            )

        # Export options
        st.download_button(
            "📥 Download JSON",
            data=json.dumps(results, indent=2),
            file_name='document_analysis.json',
            mime='application/json'
        )
```

Task 5.2: Add Advanced Features (4 hours)

- [ ] Batch processing
- [ ] History view
- [ ] Statistics dashboard
- [ ] Settings page

Day 3: Deployment

Task 5.3: Containerize Application (3 hours)

- [ ] Create Dockerfile
- [ ] Build container
- [ ] Test locally

- [ ] Push to ECR

Dockerfile:

```dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 8501

CMD ["streamlit", "run", "app.py"]
```

Task 5.4: Deploy to Production (3 hours)

- [ ] Deploy to AWS (ECS/EC2/Streamlit Cloud)
- [ ] Configure domain
- [ ] Setup HTTPS
- [ ] Test production deployment

Day 4-5: Documentation & Blog Post

Task 5.5: Write Technical Blog Post (6 hours)

- [ ] Introduction & problem statement
- [ ] Data analysis section
- [ ] Model development
- [ ] Results & evaluation
- [ ] Deployment architecture
- [ ] Lessons learned
- [ ] Future improvements

Structure: `docs/blog_post.md`

Task 5.6: Complete Documentation (4 hours)

- [ ] README with setup instructions
- [ ] API documentation
- [ ] Model card

- [ ] User guide
- [ ] Contributing guide

Task 5.7: Create Demo Video (2 hours)

- [ ] Record screen demo
- [ ] Add narration
- [ ] Upload to YouTube
- [ ] Add to README

Sprint 5 Deliverables Checklist

- [ ] Web application deployed
- [ ] Application accessible via URL
- [ ] All features working
- [ ] Blog post published
- [ ] Documentation complete
- [ ] Demo video created
- [ ] GitHub repo polished

📊 PROJECT COMPLETION CHECKLIST

GitHub Repository

- [ ] README.md with clear instructions
- [ ] requirements.txt
- [ ] All code properly organized
- [ ] .gitignore configured
- [ ] LICENSE file
- [ ] CONTRIBUTING.md

Technical Deliverables

- [ ] Document classifier (90%+ accuracy)
- [ ] Information extraction models
- [ ] SageMaker deployment
- [ ] API endpoints
- [ ] Web application
- [ ] Monitoring & logging

Documentation

- [ ] Blog post (technical audience)
- [ ] Model card
- [ ] API documentation
- [ ] Architecture diagrams
- [ ] Demo video

Performance Metrics

- [ ] Classification accuracy > 90%
- [ ] Extraction accuracy > 80%
- [ ] API latency < 1 second
- [ ] Uptime > 99%

💰 ESTIMATED COSTS

AWS Costs (Development)

- **SageMaker Training**: $3-5 per hour (GPU)
- **SageMaker Endpoints**: $0.10-0.30 per hour
- **S3 Storage**: $0.023 per GB
- **Lambda**: First 1M requests free
- **Total Development**: ~$50-100

AWS Costs (Production - Low Traffic)

- **SageMaker Endpoint**: ~$50/month
- **S3**: ~$5/month
- **Lambda**: ~$1/month
- **CloudWatch**: ~$5/month
- **Total Production**: ~$60-70/month

Cost Optimization Tips

- Use SageMaker Serverless Inference (pay per request)
- Stop endpoints when not demoing
- Use S3 lifecycle policies

- Leverage free tier where possible

🛠️ TOOLS & TECHNOLOGIES

Core Stack

- **Language**: Python 3.9+
- **ML Framework**: PyTorch 2.0
- **Transformers**: HuggingFace Transformers
- **Cloud**: AWS (SageMaker, Lambda, S3)
- **Web**: Streamlit
- **OCR**: Tesseract / AWS Textract

Development Tools

- **IDE**: VS Code / PyCharm
- **Notebooks**: Jupyter Lab
- **Version Control**: Git / GitHub
- **Container**: Docker
- **CI/CD**: GitHub Actions (optional)

Libraries

```
torch==2.0.1
transformers==4.35.0
datasets==2.14.0
sagemaker==2.180.0
boto3==1.28.0
streamlit==1.28.0
pytesseract==0.3.10
opencv-python==4.8.0
pandas==2.1.0
numpy==1.24.0
scikit-learn==1.3.0
matplotlib==3.7.0
seaborn==0.12.0
```

📈 SUCCESS CRITERIA

MVP (Minimum Viable Product)

- ✅ Classifies 4 document types with 85%+ accuracy
- ✅ Extracts key fields with 75%+ accuracy
- ✅ Working web interface
- ✅ Deployed to cloud
- ✅ Basic documentation

Target Product

- ✅ Classifies 4 document types with 90%+ accuracy
- ✅ Extracts key fields with 80%+ accuracy
- ✅ Polished web interface
- ✅ Production-ready deployment
- ✅ Comprehensive documentation
- ✅ Technical blog post

Stretch Goals

- 🎯 Support 8+ document types
- 🎯 Real-time processing
- 🎯 Multi-language support
- 🎯 Mobile app
- 🎯 API for third-party integrations

🎓 LEARNING OUTCOMES

By completing this project, you will learn:

1. End-to-end ML project lifecycle
2. Document AI & computer vision techniques
3. AWS SageMaker deployment
4. Production ML system design
5. API development & integration
6. Technical writing & communication

📞 SUPPORT & RESOURCES

Documentation

- HuggingFace: https://huggingface.co/docs
- AWS SageMaker: https://docs.aws.amazon.com/sagemaker
- PyTorch: https://pytorch.org/docs

Communities

- HuggingFace Forums
- AWS ML Community
- Stack Overflow

Sample Projects

- Udacity ML Examples
- HuggingFace Spaces
- AWS Samples GitHub

🚨 TROUBLESHOOTING

Common Issues

1. **Out of memory**: Reduce batch size
2. **Slow training**: Use smaller model or GPU
3. **Low accuracy**: More data augmentation
4. **API timeout**: Optimize inference code
5. **High costs**: Use Serverless Inference

Ready to start? Let's begin with Sprint 0!