PyCharm Safety Detection Implementation Guide

Project Setup

1. Create New PyCharm Project

```bash
# Create project directory
mkdir safety_detection_hackathon
cd safety_detection_hackathon
```

2. Install Required Libraries

```bash
# In PyCharm terminal
pip install torch torchvision opencv-python pillow numpy matplotlib
pip install ultralytics  # For YOLO
pip install roboflow     # For dataset management
pip install streamlit    # For demo app
pip install requests beautifulsoup4  # For web scraping
```

Project Structure

```
safety_detection_hackathon/
├── data/
│   ├── helmet/
│   ├── fire/
│   └── spark/
├── models/
├── src/
│   ├── dataset_creator.py
│   ├── data_preprocessor.py
│   ├── model_trainer.py
│   ├── detector.py
│   └── demo_app.py
├── results/
└── requirements.txt
```

Step 1: Dataset Creation Script

Create `src/dataset_creator.py`:

```python
```

```python
import os
import requests
import cv2
import numpy as np
from PIL import Image
import json
from pathlib import Path

class SafetyDatasetCreator:
    def __init__(self, base_dir="data"):
        self.base_dir = Path(base_dir)
        self.categories = ["helmet", "fire", "spark"]

        # Create directories
        for category in self.categories:
            (self.base_dir / category / "images").mkdir(parents=True, exist_ok=True)
            (self.base_dir / category / "labels").mkdir(parents=True, exist_ok=True)

    def create_synthetic_helmet_data(self, num_images=200):
        """Create synthetic helmet detection data"""
        print("Creating synthetic helmet detection data...")

        for i in range(num_images):
            # Create base image (construction site-like)
            img = np.random.randint(50, 200, (480, 640, 3), dtype=np.uint8)

            # Add helmet-like shapes
            num_helmets = np.random.randint(1, 4)
            labels = []

            for j in range(num_helmets):
                # Random helmet position
                x = np.random.randint(50, 590)
                y = np.random.randint(50, 430)
                w, h = np.random.randint(40, 80), np.random.randint(30, 60)

                # Draw helmet-like shape
                color = [255, 255, 0] if np.random.random() > 0.3 else [255, 255, 255]  # Yellow or white helmets
                cv2.ellipse(img, (x, y), (w//2, h//2), 0, 0, 180, color, -1)

                # YOLO format: class_id center_x center_y width height (normalized)
                center_x = x / 640
                center_y = y / 480
                norm_w = w / 640
                norm_h = h / 480
```

```
            labels.append(f"0 {center_x:.6f} {center_y:.6f} {norm_w:.6f} {norm_h:.6f}")

        # Save image and label
        img_path = self.base_dir / "helmet" / "images" / f"helmet_{i:04d}.jpg"
        label_path = self.base_dir / "helmet" / "labels" / f"helmet_{i:04d}.txt"

        cv2.imwrite(str(img_path), img)

        with open(label_path, 'w') as f:
            f.write('\n'.join(labels))

    print(f"Created {num_images} synthetic helmet images")

def create_synthetic_fire_data(self, num_images=200):
    """Create synthetic fire detection data"""
    print("Creating synthetic fire detection data...")

    for i in range(num_images):
        # Create dark base image
        img = np.random.randint(10, 80, (480, 640, 3), dtype=np.uint8)

        # Add fire-like regions
        num_fires = np.random.randint(1, 3)
        labels = []

        for j in range(num_fires):
            # Random fire position
            x = np.random.randint(50, 590)
            y = np.random.randint(50, 430)
            w, h = np.random.randint(60, 120), np.random.randint(80, 150)

            # Create fire-like gradient
            for dy in range(h):
                for dx in range(w):
                    if dx + x < 640 and dy + y < 480:
                        distance = np.sqrt(dx**2 + (dy - h*0.7)**2)
                        if distance < w/2:
                            intensity = max(0, 1 - distance/(w/2))
                            img[y+dy, x+dx] = [
                                min(255, int(intensity * 255)),      # Red
                                min(255, int(intensity * 200)),      # Green
                                min(50, int(intensity * 100))        # Blue
                            ]

            # YOLO format
            center_x = (x + w/2) / 640
            center_y = (y + h/2) / 480
```

```
            norm_w = w / 640
            norm_h = h / 480

            labels.append(f"1 {center_x:.6f} {center_y:.6f} {norm_w:.6f} {norm_h:.6f}")

        # Save image and label
        img_path = self.base_dir / "fire" / "images" / f"fire_{i:04d}.jpg"
        label_path = self.base_dir / "fire" / "labels" / f"fire_{i:04d}.txt"

        cv2.imwrite(str(img_path), img)

        with open(label_path, 'w') as f:
            f.write('\n'.join(labels))

    print(f"Created {num_images} synthetic fire images")

def create_synthetic_spark_data(self, num_images=200):
    """Create synthetic spark detection data"""
    print("Creating synthetic spark detection data...")

    for i in range(num_images):
        # Create dark industrial-like base
        img = np.random.randint(20, 100, (480, 640, 3), dtype=np.uint8)

        # Add spark-like points
        num_sparks = np.random.randint(5, 20)
        labels = []

        spark_regions = []
        for j in range(np.random.randint(1, 3)):  # 1-2 spark regions
            center_x = np.random.randint(100, 540)
            center_y = np.random.randint(100, 380)
            region_size = np.random.randint(80, 150)

            spark_count = 0
            for k in range(num_sparks):
                # Random position around center
                angle = np.random.random() * 2 * np.pi
                distance = np.random.random() * region_size

                x = int(center_x + distance * np.cos(angle))
                y = int(center_y + distance * np.sin(angle))

                if 0 < x < 640 and 0 < y < 480:
                    # Draw bright spark point
                    spark_size = np.random.randint(2, 6)
                    cv2.circle(img, (x, y), spark_size, (255, 255, 255), -1)
```

```
                cv2.circle(img, (x, y), spark_size+2, (100, 200, 255), 2)
                spark_count += 1

            if spark_count > 5:  # Only label regions with enough sparks
                # YOLO format for the spark region
                norm_x = center_x / 640
                norm_y = center_y / 480
                norm_w = region_size / 640
                norm_h = region_size / 480

                labels.append(f"2 {norm_x:.6f} {norm_y:.6f} {norm_w:.6f} {norm_h:.6f}")

        # Save image and label
        img_path = self.base_dir / "spark" / "images" / f"spark_{i:04d}.jpg"
        label_path = self.base_dir / "spark" / "labels" / f"spark_{i:04d}.txt"

        cv2.imwrite(str(img_path), img)

        if labels:
            with open(label_path, 'w') as f:
                f.write('\n'.join(labels))
        else:
            # Create empty label file
            open(label_path, 'w').close()

    print(f"Created {num_images} synthetic spark images")

def create_dataset_yaml(self):
    """Create YOLO dataset configuration"""
    yaml_content = f"""
# Safety Detection Dataset
path: {self.base_dir.absolute()}
train: images
val: images

# Classes
nc: 3
names: ['helmet', 'fire', 'spark']
"""
    with open(self.base_dir / "dataset.yaml", 'w') as f:
        f.write(yaml_content.strip())

    print("Created dataset.yaml configuration")

def create_all_datasets(self):
    """Create all synthetic datasets"""
    self.create_synthetic_helmet_data(200)
```

```
        self.create_synthetic_fire_data(200)
        self.create_synthetic_spark_data(200)
        self.create_dataset_yaml()
        print("All datasets created successfully!")

if __name__ == "__main__":
    creator = SafetyDatasetCreator()
    creator.create_all_datasets()
```

Step 2: Data Preprocessing Script

Create `src/data_preprocessor.py`:

```
python
```

```python
import os
import shutil
import random
from pathlib import Path
import cv2
import numpy as np

class DataPreprocessor:
    def __init__(self, data_dir="data", output_dir="processed_data"):
        self.data_dir = Path(data_dir)
        self.output_dir = Path(output_dir)

    def combine_and_split_data(self, train_ratio=0.8):
        """Combine all categories and split into train/val"""

        # Create output directories
        (self.output_dir / "images" / "train").mkdir(parents=True, exist_ok=True)
        (self.output_dir / "images" / "val").mkdir(parents=True, exist_ok=True)
        (self.output_dir / "labels" / "train").mkdir(parents=True, exist_ok=True)
        (self.output_dir / "labels" / "val").mkdir(parents=True, exist_ok=True)

        all_files = []

        # Collect all image files
        for category in ["helmet", "fire", "spark"]:
            img_dir = self.data_dir / category / "images"
            label_dir = self.data_dir / category / "labels"

            for img_file in img_dir.glob("*.jpg"):
                label_file = label_dir / f"{img_file.stem}.txt"
                if label_file.exists():
                    all_files.append((img_file, label_file))

        # Shuffle and split
        random.shuffle(all_files)
        split_idx = int(len(all_files) * train_ratio)

        train_files = all_files[:split_idx]
        val_files = all_files[split_idx:]

        # Copy files
        for i, (img_file, label_file) in enumerate(train_files):
            new_img_name = f"train_{i:04d}.jpg"
            new_label_name = f"train_{i:04d}.txt"

            shutil.copy2(img_file, self.output_dir / "images" / "train" / new_img_name)
```

```
            shutil.copy2(label_file, self.output_dir / "labels" / "train" / new_label_name)

        for i, (img_file, label_file) in enumerate(val_files):
            new_img_name = f"val_{i:04d}.jpg"
            new_label_name = f"val_{i:04d}.txt"

            shutil.copy2(img_file, self.output_dir / "images" / "val" / new_img_name)
            shutil.copy2(label_file, self.output_dir / "labels" / "val" / new_label_name)

        print(f"Split data: {len(train_files)} train, {len(val_files)} validation")

        # Create updated dataset.yaml
        yaml_content = f"""
path: {self.output_dir.absolute()}
train: images/train
val: images/val

nc: 3
names: ['helmet', 'fire', 'spark']
"""
        with open(self.output_dir / "dataset.yaml", 'w') as f:
            f.write(yaml_content.strip())

    def augment_data(self):
        """Apply data augmentation to training images"""
        train_img_dir = self.output_dir / "images" / "train"
        train_label_dir = self.output_dir / "labels" / "train"

        original_files = list(train_img_dir.glob("*.jpg"))

        for img_file in original_files[:50]:  # Augment first 50 images
            img = cv2.imread(str(img_file))
            label_file = train_label_dir / f"{img_file.stem}.txt"

            if not label_file.exists():
                continue

            with open(label_file, 'r') as f:
                labels = f.read().strip().split('\n')

            # Horizontal flip
            flipped_img = cv2.flip(img, 1)
            flipped_labels = []

            for label in labels:
                if label.strip():
                    parts = label.split()
```

```
                class_id, center_x, center_y, width, height = parts
                # Flip x coordinate
                new_center_x = 1.0 - float(center_x)
                flipped_labels.append(f"{class_id} {new_center_x:.6f} {center_y} {width} {height}")

            # Save augmented image and labels
            aug_img_name = f"{img_file.stem}_flip.jpg"
            aug_label_name = f"{img_file.stem}_flip.txt"

            cv2.imwrite(str(train_img_dir / aug_img_name), flipped_img)

            with open(train_label_dir / aug_label_name, 'w') as f:
                f.write('\n'.join(flipped_labels))

        print("Data augmentation completed")

if __name__ == "__main__":
    preprocessor = DataPreprocessor()
    preprocessor.combine_and_split_data()
    preprocessor.augment_data()
```

Step 3: Model Training Script

Create `src/model_trainer.py`:

python

```python
from ultralytics import YOLO
import torch
from pathlib import Path

class SafetyDetectionTrainer:
    def __init__(self, data_config="processed_data/dataset.yaml"):
        self.data_config = data_config
        self.model_dir = Path("models")
        self.model_dir.mkdir(exist_ok=True)

    def train_model(self, epochs=50, img_size=640, batch_size=16):
        """Train YOLO model for safety detection"""

        print("Starting model training...")
        print(f"Using device: {'cuda' if torch.cuda.is_available() else 'cpu'}")

        # Load pre-trained YOLOv8 model
        model = YOLO('yolov8n.pt')  # nano version for faster training

        # Train the model
        results = model.train(
            data=self.data_config,
            epochs=epochs,
            imgsz=img_size,
            batch=batch_size,
            name='safety_detection',
            patience=10,
            save=True,
            plots=True
        )

        # Save the best model
        best_model_path = self.model_dir / "best_safety_model.pt"
        model.save(str(best_model_path))

        print(f"Training completed! Best model saved to: {best_model_path}")
        return results

    def validate_model(self, model_path="models/best_safety_model.pt"):
        """Validate the trained model"""
        model = YOLO(model_path)

        # Run validation
        metrics = model.val(data=self.data_config)

        print("Validation Results:")
```

```
        print(f"mAP50: {metrics.box.map50:.4f}")
        print(f"mAP50-95: {metrics.box.map:.4f}")

        return metrics

if __name__ == "__main__":
    trainer = SafetyDetectionTrainer()

    # Train model
    results = trainer.train_model(epochs=30, batch_size=8)  # Reduced for faster training

    # Validate model
    trainer.validate_model()
```

Step 4: Detection Script

Create `src/detector.py`:

python

```python
from ultralytics import YOLO
import cv2
import numpy as np
from pathlib import Path

class SafetyDetector:
    def __init__(self, model_path="models/best_safety_model.pt"):
        self.model = YOLO(model_path)
        self.class_names = ['helmet', 'fire', 'spark']
        self.colors = {
            'helmet': (0, 255, 0),    # Green
            'fire': (0, 0, 255),      # Red
            'spark': (255, 255, 0)    # Yellow
        }

    def detect_image(self, image_path, conf_threshold=0.5):
        """Detect safety objects in an image"""

        # Run inference
        results = self.model(image_path, conf=conf_threshold)

        # Load image
        img = cv2.imread(str(image_path))

        # Process results
        for result in results:
            boxes = result.boxes
            if boxes is not None:
                for box in boxes:
                    # Get box coordinates
                    x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                    conf = box.conf[0].cpu().numpy()
                    class_id = int(box.cls[0].cpu().numpy())

                    # Get class name and color
                    class_name = self.class_names[class_id]
                    color = self.colors[class_name]

                    # Draw bounding box
                    cv2.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), color, 2)

                    # Draw label
                    label = f"{class_name}: {conf:.2f}"
                    cv2.putText(img, label, (int(x1), int(y1)-10),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

```
        return img

    def detect_video(self, video_path, output_path="output_video.mp4"):
        """Detect safety objects in a video"""

        cap = cv2.VideoCapture(str(video_path))

        # Get video properties
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        # Define codec and create VideoWriter
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # Run detection on frame
            results = self.model(frame, conf=0.5)

            # Process results
            for result in results:
                boxes = result.boxes
                if boxes is not None:
                    for box in boxes:
                        x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                        conf = box.conf[0].cpu().numpy()
                        class_id = int(box.cls[0].cpu().numpy())

                        class_name = self.class_names[class_id]
                        color = self.colors[class_name]

                        cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 2)

                        label = f"{class_name}: {conf:.2f}"
                        cv2.putText(frame, label, (int(x1), int(y1)-10),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

            # Write frame
            out.write(frame)

        # Release everything
        cap.release()
```

```
        out.release()
        cv2.destroyAllWindows()

        print(f"Video processing completed: {output_path}")

if __name__ == "__main__":
    detector = SafetyDetector()

    # Test with an image
    test_img = detector.detect_image("processed_data/images/val/val_0001.jpg")
    cv2.imshow("Detection Result", test_img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Step 5: Demo App

Create `src/demo_app.py`:

```
python
```

```python
import streamlit as st
import cv2
import numpy as np
from PIL import Image
import tempfile
import os
from detector import SafetyDetector

st.set_page_config(page_title="Safety Detection System", page_icon="🦺", layout="wide")

@st.cache_resource
def load_detector():
    return SafetyDetector("models/best_safety_model.pt")

def main():
    st.title("🦺 Safety & Surveillance Detection System")
    st.write("Upload an image to detect helmets, fire, and electrical sparks")

    # Load detector
    try:
        detector = load_detector()
    except:
        st.error("Model not found! Please train the model first using model_trainer.py")
        return

    # Sidebar
    st.sidebar.title("Settings")
    confidence = st.sidebar.slider("Confidence Threshold", 0.1, 1.0, 0.5, 0.1)

    # File upload
    uploaded_file = st.file_uploader("Choose an image", type=['jpg', 'jpeg', 'png'])

    if uploaded_file is not None:
        # Display original image
        image = Image.open(uploaded_file)

        col1, col2 = st.columns(2)

        with col1:
            st.subheader("Original Image")
            st.image(image, use_column_width=True)

        # Save uploaded file temporarily
        with tempfile.NamedTemporaryFile(delete=False, suffix='.jpg') as tmp_file:
            image.save(tmp_file.name)
```

```
            # Run detection
            with st.spinner("Detecting safety objects..."):
                result_img = detector.detect_image(tmp_file.name, conf_threshold=confidence)

            # Display result
            with col2:
                st.subheader("Detection Results")
                st.image(cv2.cvtColor(result_img, cv2.COLOR_BGR2RGB), use_column_width=True)

            # Clean up
            os.unlink(tmp_file.name)

    # Instructions
    st.markdown("""
    ## How to use:
    1. **Train the model**: Run `python src/model_trainer.py` first
    2. **Upload an image**: Use the file uploader above
    3. **Adjust confidence**: Use the slider in the sidebar
    4. **View results**: Detection boxes will appear on the right

    ## Detection Classes:
    - 🦺 **Helmet** (Green boxes): Safety helmets on construction workers
    - 🔥 **Fire** (Red boxes): Fire or flames in electrical equipment
    - ⚡ **Spark** (Yellow boxes): Electrical sparks or arcing
    """)

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

Step 6: Requirements File

Create `requirements.txt`:

```
torch>=2.0.0
torchvision>=0.15.0
ultralytics>=8.0.0
opencv-python>=4.8.0
pillow>=9.5.0
numpy>=1.24.0
matplotlib>=3.7.0
streamlit>=1.28.0
requests>=2.31.0
beautifulsoup4>=4.12.0
```

Running Instructions

1. Setup Environment

```bash
# In PyCharm terminal
pip install -r requirements.txt
```

2. Create Dataset

```bash
python src/dataset_creator.py
```

3. Preprocess Data

```bash
python src/data_preprocessor.py
```

4. Train Model

```bash
python src/model_trainer.py
```

5. Test Detection

```bash
python src/detector.py
```

6. Run Demo App

```bash
streamlit run src/demo_app.py
```

Expected Timeline

- **Dataset Creation**: 30 minutes
- **Model Training**: 2-4 hours (depending on hardware)
- **Testing & Demo**: 30 minutes
- **Total**: ~4-5 hours

Tips for Success

1. **Start Simple**: Begin with helmet detection only, add others if time permits
2. **Use GPU**: Training will be much faster with CUDA-enabled GPU
3. **Monitor Training**: Watch loss curves to avoid overfitting
4. **Test Early**: Validate on sample images during training
5. **Backup Plan**: Keep a simple rule-based detector as fallback

Troubleshooting

- **CUDA errors**: Use CPU training with smaller batch sizes
- **Memory issues**: Reduce image size or batch size
- **Poor detection**: Increase dataset size or training epochs
- **Slow training**: Use YOLOv8n (nano) instead of larger models

Good luck with your hackathon! 🚀