**1. Rancang Jalur Kerja (Pipeline) secara Menyeluruh**

1. **Data Acquisition (Pencarian dan Pengumpulan Data)**
   * Pastikan Anda memiliki **gambar defect** dengan resolusi yang memadai.
   * Ambil data pada **berbagai kondisi pencahayaan**, sudut pandang, variasi warna kain, dan variasi pola jahitan.
   * Usahakan **variasi data** sebanyak mungkin agar model mampu *generalize* dengan baik.
2. **Data Labeling**
   * Untuk pendekatan *object detection* (menggunakan YOLO), Anda perlu memberikan **bounding box** dan label pada area yang mengalami defect.
   * Label bisa dibagi menjadi dua kelas: broken dan pleated.
   * Gunakan alat bantu seperti [LabelImg](https://github.com/tzutalin/labelImg), [Label Studio](https://github.com/heartexlabs/label-studio), atau platform seperti [Roboflow](https://roboflow.com/) untuk memudahkan proses pelabelan.
3. **Model Training**
   * Anda sudah berpengalaman dengan YOLO berbasis PyTorch, jadi bisa langsung memanfaatkan repositori YOLOv5 atau YOLOv8.
   * Lakukan **eksperimen hyperparameter** (batch size, image size, learning rate, optimizer, dsb).
   * Gunakan **augmentasi** (flip, rotate, color jitter, random crop) agar model lebih robust.
4. **Evaluation & Validation**
   * Pakai metrik seperti mAP (mean Average Precision) untuk mengukur performa deteksi.
   * Lihat juga metrik *precision*, *recall*, dan *F1-score* untuk masing-masing kelas broken dan pleated.
5. **Deployment / Inference**
   * Setelah model terlatih, Anda bisa melakukan *inference* secara *real-time* dengan GPU, atau secara *offline* di server.
   * Integrasikan model ke alat inspeksi dengan kamera (misalnya via OpenCV) untuk menangkap feed video/gambar, lalu jalankan inference YOLO.
   * Output berupa bounding box dan label defect (broken / pleated) di area kain yang rusak.

**2. Strategi Menggunakan YOLO untuk Membedakan Broken vs Pleated**

Karena Anda ingin membedakan dua tipe defect, ada dua pendekatan utama:

1. **Single-Stage Detection + Classification**
   * Langsung membuat *custom dataset* dengan dua kelas: broken dan pleated.
   * Ketika training YOLO, masing-masing bounding box dilabeli sesuai kelasnya. Hasil deteksi otomatis mengklasifikasikan area cacat.
   * Ini adalah pendekatan paling sederhana, karena sekali inferensi YOLO langsung memberikan: bounding box + label tipe cacat.
2. **Two-Stage Approach**: Deteksi dulu, kemudian klasifikasi
   * Tahap 1 (deteksi cacat apa pun). Buat YOLO satu kelas: defect.
   * Tahap 2 (klasifikasi jenis defect). Potong area bounding box yang terdeteksi, lalu jalankan model klasifikasi CNN terpisah (misal ResNet, atau model custom).
   * Pendekatan ini berguna jika misalnya Anda ingin *fokus mendeteksi semua defect dulu*, lalu melabeli tipenya secara lebih mendalam/tepat. Tapi akan sedikit lebih kompleks dalam pipeline.

**Rekomendasi:** Gunakan **Single-Stage** karena Anda hanya memiliki dua kelas. Lebih efisien dan mudah dalam implementasi.

**3. Persiapan Dataset secara Detil**

1. **Jumlah dan Kualitas Data**
   * Idealnya, setiap kelas (baik broken maupun pleated) harus punya ratusan hingga ribuan contoh. Lebih banyak lebih baik.
   * Jika sulit mengumpulkan data asli, pertimbangkan **data augmentation** yang agresif.
2. **Variasi Kondisi Produksi**
   * Usahakan di dataset terdapat variasi material kain, warna, pola jahit, background, dan pencahayaan.
   * *Noise* umum di pabrik (bayangan dari tangan, benang lain yang tergantung, dsb) sebaiknya disertakan agar model terbiasa melihat *real environment*.
3. **Distribusi Kelas**
   * Pastikan jumlah data broken dan pleated tidak terlalu timpang. Jika timpang, gunakan teknik *oversampling* atau *undersampling* untuk mengatasi *class imbalance*.

**4. Detil Implementasi dengan YOLO (misal: YOLOv5)**

1. **Persiapan Lingkungan**
   * Install *dependencies* (PyTorch, OpenCV, dsb).
   * Clone repository YOLOv5 (atau YOLOv8) dari GitHub.
2. **Struktur Direktori Dataset**
   * YOLOv5 membutuhkan struktur seperti:

bash

Copy code

dataset/

images/

train/

val/

test/

labels/

train/

val/

test/

* + Setiap *split* (train/val/test) berisi file gambar dan file label dengan format .txt.

1. **File Konfigurasi dan Hyperparameter**
   * Buat file .yaml yang mendefinisikan daftar kelas:

yaml

Copy code

nc: 2

names: ['broken', 'pleated']

* + Pastikan path ke folder train dan val sudah benar.

1. **Training**
   * Jalankan perintah, misalnya (untuk YOLOv5):

bash

Copy code

python train.py --data data.yaml --cfg yolov5s.yaml --weights yolov5s.pt --epochs 100

* + **Monitor** *loss* dan metrik mAP, precision, recall.
  + Uji beberapa *learning rate*, *optimizer*, *image size*, dll.

1. **Evaluasi Model**
   * Setelah training, jalankan:

bash

Copy code

python val.py --weights best.pt --data data.yaml

* + Perhatikan mAP (0.5) dan (0.5:0.95), precision, recall.

1. **Inference**
   * Tes model di gambar/video:

bash

Copy code

python detect.py --weights best.pt --source path\_to\_your\_test\_images\_or\_video

* + Cek apakah bounding box dan label sudah sesuai harapan.

**5. Penggunaan OpenCV untuk Aplikasi Real-Time**

1. **Integrasi Kamera**
   * Gunakan cv2.VideoCapture(0) (atau sesuai indeks/koneksi kamera) untuk menangkap gambar secara real-time.
   * Pada loop utama, ambil frame, lalu kirim ke model YOLO untuk *inference*.
2. **Visualisasi Deteksi**
   * Dari hasil *inference*, Anda akan mendapatkan koordinat bounding box dan label prediksi (misal “broken” / “pleated”).
   * Gunakan fungsi seperti cv2.rectangle() dan cv2.putText() untuk menandai area cacat di layar.
3. **Optimasi Kecepatan**
   * Jika pemrosesan terlalu lambat, pastikan Anda menggunakan GPU. Atau turunkan resolusi input (trade-off antara kecepatan dan akurasi).
   * Bisa juga menggunakan model yang lebih ringan, seperti YOLO-nano, YOLO-tiny, dsb.

**6. Analisa dan Potensi Masalah di Lapangan**

1. **Lighting dan Kontras Gambar**
   * Cahaya yang tidak konsisten dapat membuat *defect* sulit dibedakan. Pastikan pencahayaan cukup stabil.
   * Dapat juga ditambahkan modul *illumination correction* sederhana (misal histogram equalization, dsb).
2. **Background Bervariasi**
   * Kadang background area jahitan bisa menyerupai *defect* (misal benang putus halus). Perlu dataset cukup banyak agar model belajar membedakannya.
3. **Defect yang Tersamar**
   * *Broken stitch* yang sangat halus atau *pleated* yang sangat kecil mungkin sulit dideteksi. Pastikan resolusi kamera cukup tinggi.
   * Pertimbangkan *optical zoom* jika diperlukan.
4. **Overfitting**
   * Jika dataset terlalu sedikit atau kurang variatif, model cenderung *overfit*. Gunakan *data augmentation* dan *regularization* (dropout, dsb.) sesuai kebutuhan.
5. **Kesalahan Klasifikasi**
   * Mungkin akan ada beberapa overlap: kadang “jahitan rusak” dan “lipatan kain” tampak mirip. Pastikan labeling konsisten dan perbanyak contoh kasus yang membingungkan (hard examples).

**7. Kesimpulan**

* **Single-Stage Detection** menggunakan YOLO (dengan dua kelas: broken dan pleated) adalah solusi paling efektif dan sederhana untuk kasus Anda.
* Kunci keberhasilan utama ada pada **kualitas dan kuantitas data** serta **variasi** pada dataset pelatihan.
* Gunakan **augmentasi** dan **eksperimen hyperparameter** untuk mendapatkan hasil maksimal.
* Setelah model siap, integrasikan dengan pipeline OpenCV untuk *real-time inference* di kamera pabrik.
* Jangan lupa evaluasi hasil di lapangan (dengan kain nyata di jalur produksi), karena kondisi nyata seringkali lebih rumit daripada skenario laboratorium.