1. DETECCIÓN DE ERRORES EN PRENDAS CON VISIÓN POR COMPUTADOR Y STREAMING

CONTEXTO:

• Tenemos una cámara instalada en la línea de producción textil que captura en tiempo real imágenes de las prendas mientras avanzan por la cadena de fabricación.

• Queremos detectar defectos (ej. costuras mal hechas, manchas, roturas) en dichas prendas utilizando un modelo de inteligencia artificial que aprenda a partir de ejemplos previos.

• El sistema debe operar en tiempo real (streaming) para emitir alertas inmediatas cuando se detecta una prenda defectuosa y así poder retirarla o corregirla de manera temprana.

OBJETIVO:

• Describir la arquitectura completa y el flujo de datos necesario para implementar la detección de defectos con un sistema de visión por computador en tiempo real.

• Proponer el tipo de modelo o enfoque de aprendizaje (ej. aprendizaje supervisado, redes neuronales convolucionales, transfer learning, etc.) que utilizarías y por qué.

• Explicar de qué manera gestionarías el entrenamiento constante del modelo (aprendizaje continuo) para mejorar la exactitud de la detección con el tiempo.

• Mencionar las herramientas y frameworks que emplearías (por ejemplo: TensorFlow, PyTorch, OpenCV, servicios en la nube, sistemas de mensajería en streaming, etc.).

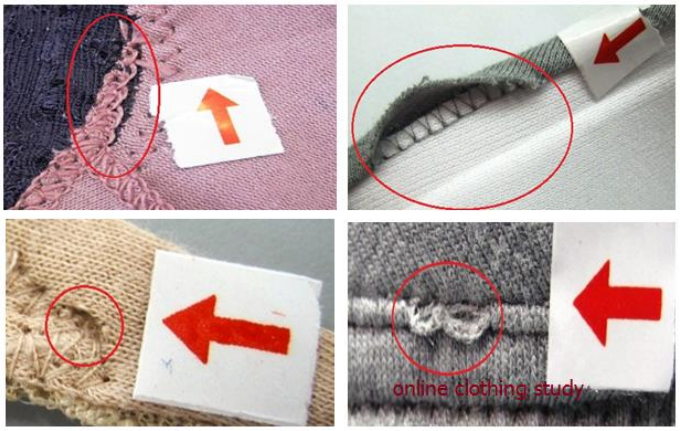
• Incluir consideraciones para la implementación y despliegue en una planta de producción, como infraestructura, latencia, escalabilidad y mantenimiento.

Propuesta para obtener el puesto de Ingeniero en IA,
presentada por Brian Buendia Sosa.

1.1. PLANTEAMIENTO

Esta aplicación busca detectar defectos en las prendas que pasan por la línea de producción. Para ello, comenzamos por considerar las características específicas de esta situación, las cuales serán de importancia para la implementación del sistema.

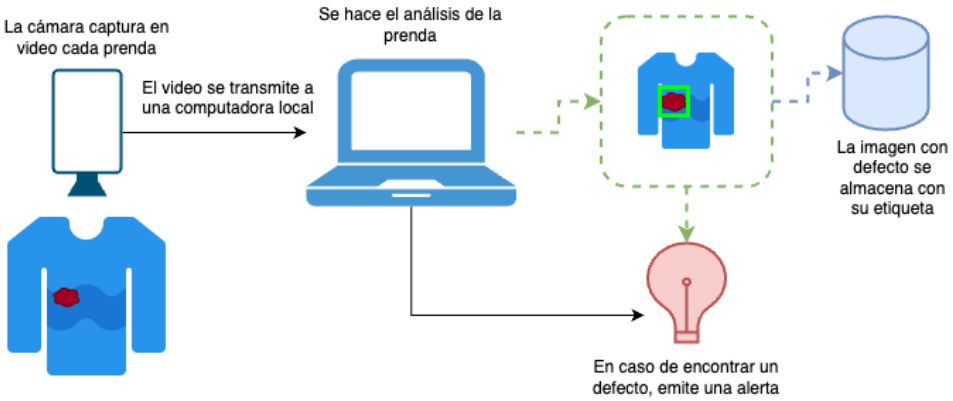
Se requiere que alguien con experiencia en el área de calidad proporcione una lista de los defectos que suelen presentarse en las prendas, ya que estos serán las categorías que nuestro modelo deberá detectar.



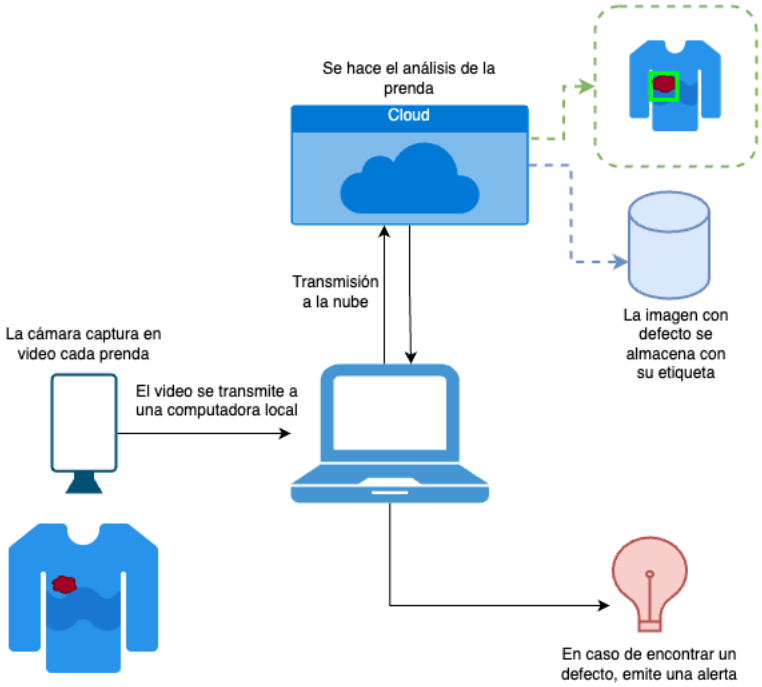
Un detalle importante a tener en cuenta es saber cómo se mueve la ropa sobre la cinta automática, la posición en la que está y cómo la cámara la va a detectar. Esto es relevante porque existe la posibilidad de que los defectos aparezcan en la parte trasera de la ropa, pero si ese lado nunca es visible, la cámara no podrá detectar defectos allí. Además, esto también se debe considerar para la obtención de datos necesarios para entrenar el modelo, ya que, en ese caso, se deben tomar fotografías de las prendas tanto de frente como por detrás.

Se pueden tomar dos enfoques para realizar el análisis del video: hacerlo de forma local en una computadora o utilizar un servicio en la nube como AWS, Azure o GCP.

En el siguiente diagrama, se muestra el flujo de datos de forma general utilizando una computadora local. Esta computadora puede ser un hardware incluido en el mismo dispositivo de la cámara, que cuenta con la capacidad para realizar análisis de video, o una computadora destinada específicamente para esta aplicación. La cámara captura el video y lo envía a la computadora, donde se lleva a cabo el análisis. Si se detecta un defecto, el sistema puede emitir una alerta para notificar al operador. Esta alerta podría ser mediante una luz, un sonido o mostrando la notificación en una pantalla.



El siguiente diagrama muestra cómo sería el flujo si se utiliza un servicio en la nube. El proceso es similar, pero, en lugar de analizar el video en un equipo local, la información se envía al servicio en la nube, donde se realiza el análisis. Posteriormente, el servicio envía una alerta de vuelta al equipo local, donde se puede mostrar una notificación de detección de defectos en las prendas.



Cada una de estas aproximaciones presenta ventajas y desventajas. Tomando en cuenta las necesidades y capacidades de la empresa, se puede optar por una u otra. La propuesta de usar una computadora local elimina las complicaciones de enviar los datos a la nube, donde se requiere una conexión con alto ancho de banda, además del preprocesamiento del video usando protocolos de compresión y transmisión a la nube para disminuir la latencia lo más posible. El inconveniente es que se necesita un equipo con una tarjeta GPU capaz de procesar en tiempo real el video y ejecutar el modelo de detección de defectos.

Por otro lado, los servicios en la nube suelen ofrecer una gran cantidad de herramientas que hacen más sencillo el despliegue y el entrenamiento del modelo, por lo que, si se desea implementar rápidamente, podría ser una buena opción. Sin embargo, también incrementan los costos al tener que pagar por el servicio.

Se contempla que se usaría una computadora de forma local para llevar a cabo esta tarea, ya que, una vez que se tenga el modelo funcionando de manera local, es más sencillo migrarlo a la nube que hacer lo contrario. Se necesita una GPU con suficiente capacidad para inferencia en tiempo real, como una RTX 3060 o superior, aunque la elección final dependerá de los requerimientos del modelo y de la tasa de imágenes procesadas por segundo.

Para llevar a cabo estas tareas, se cuenta con diversas opciones de librerías y frameworks que se pueden utilizar. Se propone usar Python como lenguaje para agilizar todo el proceso, debido a su integración fácil con otras herramientas que se van a utilizar.

Para el procesamiento de imágenes, se puede usar Pillow u OpenCV. Pillow es más ligero y sencillo, pero también tiene menos funciones, por lo que se opta por OpenCV para el preprocesamiento de imágenes. Además, OpenCV es especialmente útil para tareas de visión por computadora, ya que permite mostrar en una ventana el video capturado por la cámara y, posteriormente, dibujar los recuadros (bbox) que indican los defectos detectados.

Para implementar la arquitectura de aprendizaje profundo, existen diversas opciones, pero las más populares son TensorFlow y PyTorch. PyTorch ofrece mayor flexibilidad para editar diversos parámetros en la arquitectura, por lo que se opta por este framework.

1.2. OBTENCIÓN DE LOS DATOS

Considerando que se cuenta con una cámara en la línea de producción textil, se pueden tomar fotografías de las prendas que presentan defectos.

Es importante tomar en cuenta el contexto en el que se capturan las fotografías y si este puede variar con el tiempo. Por ejemplo, se debe considerar si el fondo sobre el que está la ropa es de cierto color y si este puede cambiar posteriormente, así como la iluminación del lugar, la cual puede variar en el transcurso del día en caso de que no esté estrictamente controlada en la línea de producción.

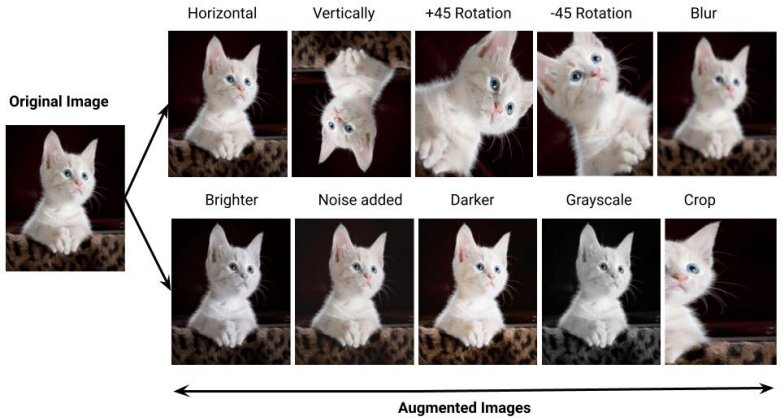
Para lograr una mayor variedad en los datos, se pueden tomar varias fotografías de las prendas, variando su posición, dobleces, etc. Es fundamental obtener múltiples imágenes de los diferentes defectos que se presentan y tratar de que el número de fotografías para cada categoría esté lo más balanceado posible.

Dado que tomar miles de fotografías, especialmente de prendas con defectos, puede ser un proceso muy demorado, se puede hacer uso de bases de datos existentes para evitar entrenar el modelo desde cero.

Las imágenes pueden almacenarse en formato .jpg para reducir el tamaño de almacenamiento.

1.3. PREPARACIÓN DE LOS DATOS

Una vez que tenemos todos los datos recopilados, podemos realizar aumentación de datos con las imágenes de las prendas que presentan defectos. Esta técnica implica modificar ligeramente las imágenes aplicando rotaciones, zoom, traslaciones o ligeros cambios de color. De esta forma, podemos incrementar significativamente el número de imágenes en nuestra base de datos.



Es recomendable hacer una copia de todos los datos recopilados para trabajar sobre ella y no sobre los datos originales, evitando así cualquier problema futuro relacionado con la pérdida de información.

Se propone utilizar la técnica de detección de objetos, lo que implica que el modelo generará una caja que rodeará al objeto en cuestión. Si es importante determinar también el tamaño del defecto o su posición en la prenda, se pueden emplear otras técnicas, como la segmentación de imágenes (*image segmentation*). Sin embargo, por ahora, consideraremos que el objetivo principal es detectar los defectos.

Para ello, es necesario realizar anotaciones en nuestras imágenes, es decir, etiquetar manualmente los defectos y encerrarlos en una caja (*bounding box*), para luego generar las coordenadas y las etiquetas de clasificación de cada defecto. Para esta tarea, se pueden usar herramientas como www.makesense.ai o www.cvat.ai.

Posteriormente, se exportan las etiquetas en formato .json, el cual contiene información sobre la imagen, la categoría del defecto y las coordenadas de la caja delimitadora.

Una vez que tengamos nuestro conjunto de datos, podemos dividirlo en los porcentajes estándar para el entrenamiento: 70% para entrenamiento, 20% para pruebas y 10% para validación.

Se propone utilizar un modelo de aprendizaje supervisado, ya que habrá una persona encargada de etiquetar manualmente los defectos en el conjunto de datos. Este enfoque es importante porque en las prendas pueden existir muchos detalles que podrían no ser claros para el modelo, y queremos evitar confusiones. Por ejemplo, es crucial que el modelo pueda distinguir entre un hoyo y el ojal de un botón, o entre una mancha de grasa y el estampado de una camiseta.

Además, se sugiere agregar a la base de datos fotografías de prendas sin defectos. Esto permitirá que las categorías incluyan los defectos detectados hasta ahora en la línea de producción, más una categoría adicional llamada “sin defecto”. Esta inclusión ayudará a que el modelo generalice mejor y, en la medida de lo posible, reduzca los falsos positivos.

1.4. EXPLORACIÓN DE DIFERENTES MODELOS

Para la detección de objetos, existen diversas opciones de arquitectura que se pueden utilizar. Dos de las más empleadas son YOLO y Faster R-CNN. Dependiendo de los requerimientos específicos del problema, una puede ser más adecuada que la otra. Si necesitamos analizar imágenes en tiempo real con un tiempo de latencia muy bajo, capaz de procesar decenas de imágenes por segundo, la mejor opción es YOLO. Sin embargo, si requerimos mayor precisión, incluso si esto implica comprometer la latencia, Faster R-CNN ofrece mejores resultados.

En primer lugar, se propone utilizar Faster R-CNN, ya que algunos defectos podrían ser muy finos y necesitamos un alto nivel de precisión. Faster R-CNN es uno de los modelos más robustos para detectar objetos pequeños, lo cual es útil en este caso, pues hay defectos en las prendas que pueden ser de tamaño reducido. Finalmente, si Faster R-CNN no cumple con los requisitos de latencia deseados, se puede evaluar la opción de implementar YOLO.

1.5. AFINACIÓN DEL MODELO

Utilizamos Faster R-CNN con un backbone de ResNet50 para lograr un equilibrio entre precisión y velocidad.

Como se mencionó anteriormente, la cantidad de datos de entrenamiento es sumamente importante. Considerando que este tipo de redes suelen entrenarse con cientos de miles o incluso millones de imágenes para obtener resultados óptimos, se opta por utilizar fine-tuning para el entrenamiento, utilizando como base los pesos del dataset COCO. Este dataset cuenta con 80 categorías, pero no está enfocado específicamente en prendas. Con el fine-tuning, se busca que las características que el modelo ya aprendió a detectar se enfoquen y especialicen en nuestro objetivo, que es la detección de defectos en ropa.

A diferencia de la mayoría de las investigaciones, que se centran en detectar defectos en tela, nuestro contexto es distinto, ya que las prendas tienen muchos detalles, como distintos tipos de prendas, colores, tallas, estampados, botones, etc. Por esto, se propone realizar primero un fine-tuning para que el modelo aprenda a detectar distintos tipos de prendas. Esto ayudará a que generalice mejor el reconocimiento de diversas prendas y no se confunda cuando aparezcan nuevos modelos.

Para este primer fine-tuning, se propone utilizar el dataset Fashion Product Images, el cual consta de 44,000 datos etiquetados. Este dataset incluye prendas de distintas categorías, para hombres y mujeres, con diferentes colores, estilos deportivos, etc. Cuenta con un 48% de fotos de prendas, por lo que podemos filtrar solo estos datos y descartar los demás. Entre sus categorías se encuentran playeras, jeans, tops, etc.

Después de realizar este primer fine-tuning, podemos llevar a cabo el fine-tuning principal utilizando nuestra base de datos con defectos en las prendas.

El entrenamiento se realiza ajustando los hiperparámetros según las características del problema. Se podría comenzar con un batch size de 16 y un learning rate de 0.01.

Hacer fine-tuning implica modificar los pesos de las capas ocultas, lo cual puede afectar positiva o negativamente nuestro objetivo. Para mitigar esto, se pueden congelar las primeras capas del modelo, permitiendo que las características más generales (como la forma de las prendas) se mantengan intactas, mientras que las capas finales se adapten para aprender características más específicas, como los defectos en las prendas.

Dado que la última capa de clasificación de categorías utiliza la función softmax para predecir la probabilidad de que un objeto pertenezca a cierta categoría, se utiliza Stochastic Gradient Descent (SGD) como algoritmo de optimización.

Al finalizar el entrenamiento, se obtienen las métricas para cuantificar los resultados. Mean Average Precision (mAP) suele ser la métrica más utilizada en detección de objetos. Esta medida se basa en la matriz de confusión, Intersection over Union (IoU), Recall y Precision. Sin embargo, también es útil mantener estas mediciones por separado para tener una idea más clara de los falsos positivos y falsos negativos, lo que nos permite determinar si el modelo está dejando pasar defectos sin detectarlos o si está identificando detalles originales de la ropa como si fueran defectos.

Finalmente, se puede probar el modelo utilizando prendas controladas por nosotros, es decir, prendas de las cuales sabemos de antemano si tienen defectos o no, y, en caso de tenerlos, qué tipo de defectos son y dónde se encuentran. De esta manera, podemos evaluar el desempeño del modelo en un entorno más realista y ajustarlo según sea necesario.

1.6. LANZAMIENTO Y MONITOREO

Dado que ya se cuenta con una cámara industrial instalada, se asume que también se dispone del hardware al cual esta cámara está adaptada. Para implementar el modelo, se utiliza este hardware, el cual tiene control sobre la cámara. Esta implementación es preferible a optar por un servicio en la nube, ya que la latencia disminuye considerablemente y no se depende de factores externos, como la velocidad de internet.

El modelo se implementa en esta computadora y se lee la información de la cámara. En cada frame, el modelo realiza el análisis de la imagen y detecta los defectos en tiempo real, encerrándolos en una bounding box (bbox) y etiquetando la categoría del defecto.

Cuando se detecta un defecto, se puede activar una alerta visual o sonora, como una pequeña luz parpadeante o un sonido, para notificar al operario y que este retire la prenda con defectos.

Se propone que, cada vez que se detecte una prenda con defectos, la imagen se almacene automáticamente en la memoria. Periódicamente, dependiendo de la cantidad de prendas procesadas por día, estas imágenes se revisan manualmente para verificar que no haya errores en las detecciones. Posteriormente, estas imágenes se utilizan para realizar un entrenamiento continuo del sistema. Además, es importante obtener las métricas correspondientes para asegurarse de que el modelo siga generalizando correctamente.

Para evitar problemas como el olvido catastrófico, que suele ocurrir al entrenar periódicamente el modelo con nuevos datos, se proponen algunas técnicas, como utilizar imágenes de entrenamientos anteriores junto con el nuevo dataset, emplear un learning rate bajo y congelar algunas de las capas ocultas del modelo.

La periodicidad del reentrenamiento depende del volumen de producción. Es necesario realizar un análisis para determinar cuántas prendas se analizan por día y cuántas de estas presentan defectos. Si la cantidad de imágenes con defectos es muy pequeña (por ejemplo, menos de 100), podría no ser conveniente realizar el reentrenamiento hasta acumular un número mayor, como 100 o más imágenes.