PROCESAMIENTO DE IMAGEN Y SEÑALES

INFORME TÉCNICO: PROYECTO FINAL

SEGMENTACIÓN DE ÁREA DE INTERÉS Y K-MEANS EN IMÁGENES IR

INTEGRANTES GRUPO 5:

CESAR NELSON ABARCA ARAUJO

JHONNY NICOLAS RAMIREZ BARAHONA

TEÓFILO MANUEL CHÓEZ ARTEAGA

Contenido

1. Introducción

El presente informe detalla el desarrollo del proyecto final, el cual aborda la segmentación automática del área de interés (zona de soldadura) en imágenes infrarrojas (IR) y la posterior aplicación del algoritmo de clustering K-Means sobre la región extraída (ROI). El objetivo principal es separar las zonas calientes del fondo térmico mediante un criterio sólido y automático, para luego agrupar los píxeles de interés de manera eficiente, evaluando la mejora en los tiempos de procesamiento computacional

2. Metodología

El flujo de trabajo se implementó en Python empleando las librerías OpenCV, NumPy y Scikit-learn. Se procesó un dataset total de 347 imágenes térmicas en formato .tif, el cual se dividió en un conjunto de entrenamiento (277 imágenes) y un conjunto de prueba (70 imágenes). La metodología se dividió en dos fases:

- Fase 1 - Segmentación (Técnica Seleccionada): Se utilizó la Umbralización de Otsu para calcular automáticamente el mejor umbral y separar la zona caliente. Posteriormente, se aplicaron operaciones morfológicas: Opening (erosión + dilatación) para eliminar el ruido (pequeños puntos blancos) y Closing (dilatación + erosión) para rellenar los huecos dentro de la región de interés.
- Fase 2 - Clustering: Se aplicó el algoritmo K-Means estrictamente sobre los píxeles extraídos por la máscara generada (ROI), utilizando k=4 clusters empíricos

3. Validación Visual y Cuantitativa del Test Set

La técnica de segmentación (Umbralización + Morfología) se aplicó exitosamente al 100% de las imágenes del conjunto de prueba (70 imágenes) para validar su generalización. El análisis cuantitativo demostró gran consistencia: las áreas segmentadas (la zona real de soldadura) representaron de manera estable entre el 0.38% y el 2.23% del total de la imagen original. Esto confirmó una reducción significativa de la dimensionalidad de los datos sin perder información térmica relevante

4. Aplicación de K-Means en el Área de Interés (ROI)

Durante el entrenamiento del algoritmo K-Means sobre el ROI de las 277 imágenes del Trainset, se extrajeron y procesaron un total de 2,830,112 píxeles de interés.

- Distribución Térmica (Centroides): Los cuatro clusters identificaron intensidades medias de 81.57, 93.35, 105.20 y 119.47.
- Rendimiento: El algoritmo K-Means se completó en apenas 14.86 segundos (19.55s incluyendo la extracción del ROI).

5. Discusión de Resultados y Comparativa de Rendimiento

Al responder las interrogantes planteadas en el proyecto, se establecen los siguientes puntos:

- Idoneidad de los Filtros: Los criterios fueron altamente adecuados. La umbralización de Otsu aprovecha el claro contraste de la zona caliente para automatizar el umbral, mientras que la morfología estabiliza los bordes difusos. Esto evitó ajustes manuales por imagen y logró el objetivo de forma robusta.
- Optimización del Tiempo de Entrenamiento: El tiempo de ejecución se redujo de manera drástica frente al análisis de la imagen completa. Mientras que la imagen completa requiere procesar más de 85,094,400píxeles por archivo,

el uso del ROI redujo este volumen a unos ~4,000 píxeles en promedio por imagen. Esta focalización de datos permitió que el algoritmo convergiera mucho más rápido y optimizara el consumo de memoria

6. Áreas de Mejora Futura

Si bien el resultado fue exitoso, el sistema podría perfeccionarse implementando las siguientes medidas:

- Emplear métricas formales (como el método del codo o índice de silueta) para justificar la elección óptima del número de clusters (k).
- Aplicar ecualización adaptativa (CLAHE) como preprocesamiento para normalizar variaciones ambientales o de calibración del sensor.
- Otorgar una interpretación física a los clusters para traducir la intensidad del píxel a rangos de temperatura reales (zona de precalentamiento, punto de fusión, etc.) y avanzar hacia la detección automática de defectos industriales (anomalías térmicas).

7. Conclusiones Generales

El proyecto demuestra de forma contundente que segmentar previamente un área de interés (ROI) limpia el ruido irrelevante, reduce el volumen de información y mejora notablemente la eficiencia y velocidad de algoritmos de Machine Learning como K-Means. La combinación de Otsu con limpieza morfológica generó clusters más representativos que fueron aplicados con éxito al Testset, perfilando esta metodología como altamente viable para entornos de producción real, como la inspección automática y monitoreo de calidad de soldaduras térmicas.

8. Referencias al Código

Las implementaciones principales de este flujo de trabajo recaen en las siguientes funciones extraídas del cuaderno de Python:

- Segmentación: Uso de cv2.threshold con cv2.THRESH_OTSU para la separación térmica, seguido de cv2.morphologyEx con cv2.MORPH_OPEN y cv2.MORPH_CLOSE para la limpieza.
- Extracción ROI: Aplicación de la máscara sobre la imagen usando cv2.bitwise_and.
- Clustering: Implementación de cv2.kmeans limitando el algoritmo exclusivamente a la matriz de datos pixel_values extraída de las máscaras

9. Anexo: Implementación de Código Fuente

Las siguientes funciones de Python utilizan las librerías OpenCV (cv2) y NumPy (np) para llevar a cabo las dos fases principales del proyecto: la segmentación del área de interés (ROI) y el agrupamiento (Clustering) mediante K-Means.

9.1. Fase 1: Segmentación (Umbralización de Otsu + Morfología)

Esta función se encarga de separar automáticamente la zona caliente de la soldadura y aplicar filtros morfológicos para limpiar el ruido y rellenar los huecos en la región segmentada

```
def tecnica_1_umbral_morfologia(img, visualizar=True):
    """
    Técnica 1: Umbralización adaptativa + Morfología

    Pasos:
    1. Umbralización de Otsu para separar zona caliente
    2. Morfología Opening (eliminar ruido)
    3. Morfología Closing (rellenar huecos)
    """
    # Paso 1: Umbralización de Otsu
    # Otsu calcula automáticamente el mejor umbral
    ret, thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    print(f"Umbral de Otsu calculado: {ret:.2f}")

    # Paso 2: Morfología - Opening (Erosión + Dilatación)
    # Elimina puntos blancos pequeños (ruido)
    kernel_small = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel_small,
iterations=2)

    # Paso 3: Morfología - Closing (Dilatación + Erosión)
    # Rellena agujeros negros pequeños
    kernel_large = np.ones((5, 5), np.uint8)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel_large,
iterations=2)

    # Máscara final
    mask_t1 = closing
```

```
    if visualizar:
        plt.figure(figsize=(18, 8))

        plt.subplot(2, 3, 1)
        plt.imshow(img, cmap='gray')
        plt.title('Original')
        plt.axis('off')

        plt.subplot(2, 3, 2)
        plt.imshow(thresh, cmap='gray')
        plt.title(f'Otsu (umbral={ret:.1f})')
        plt.axis('off')

        plt.subplot(2, 3, 3)
        plt.imshow(opening, cmap='gray')
        plt.title('Opening (elimina ruido)')
        plt.axis('off')

        plt.subplot(2, 3, 4)
        plt.imshow(closing, cmap='gray')
        plt.title('Closing (rellena huecos)')
        plt.axis('off')

        # Aplicar máscara a imagen original
        resultado = cv2.bitwise_and(img, img, mask=mask_t1)

        plt.subplot(2, 3, 5)
        plt.imshow(resultado, cmap='hot')
        plt.title('Área de Interés Extraída')
        plt.colorbar()
        plt.axis('off')

        plt.subplot(2, 3, 6)
        plt.imshow(mask_t1, cmap='gray')
        plt.title('Máscara Final (Técnica 1)')
        plt.axis('off')

        plt.tight_layout()
        plt.show()

    return mask_t1

# Ejecutar Técnica 1
print("=" * 60)
```

```
print("TÉCNICA 1: Umbralización + Morfología")
print("=" * 60)
mask_1 = tecnica_1_umbral_morfologia(img)
```

9.2. Fase 2: Extracción del ROI y Algoritmo K-Means

Esta función recorre el conjunto de imágenes de entrenamiento, extrae estrictamente los píxeles correspondientes al Área de Interés (usando la máscara generada en el paso anterior) y entrena el modelo K-Means con un número predefinido de clusters (k=4).

```
def kmeans_en_roi(train_dir, n_clusters=4):
    """
    Aplica K-means SOLO a los píxeles del ROI de todas las imágenes del
trainset.

    Args:
        train_dir: Directorio con imágenes de entrenamiento
        n_clusters: Número de clusters (3 o 4)

    Returns:
        centers_sorted: Centroides ordenados
        tiempo_total: Tiempo de ejecución en segundos
        estadisticas: Dict con stats de cada cluster
    """
    print(f"\n{'='*70}")
    print(f"K-MEANS EN ROI - ENTRENAMIENTO (n_clusters={n_clusters})")
    print(f"{'='*70}\n")

    inicio_total = time.time()

    # Paso 1: Extraer píxeles del ROI de todas las imágenes
    images = list(Path(train_dir).glob('*.tif'))
    all_roi_pixels = []

    print(f"📂 Procesando {len(images)} imágenes del trainset...\n")

    for idx, img_path in enumerate(images):
        # Cargar imagen
        img = cv2.imread(str(img_path), cv2.IMREAD_GRAYSCALE)
        if img is None:
            continue
```

```
        # Extraer ROI
        mask = tecnica_1_umbral_morfologia(img, visualizar=False)

        # Obtener solo píxeles del ROI
        roi_pixels = img[mask > 0]
        all_roi_pixels.append(roi_pixels)

        if (idx + 1) % 50 == 0:
            print(f"   [{idx+1}/{len(images)}] Procesadas...
({len(roi_pixels)} píxeles ROI)")

    # Paso 2: Concatenar todos los píxeles del ROI
    pixel_values = np.concatenate(all_roi_pixels).reshape(-1, 1)
    pixel_values = np.float32(pixel_values)

    print(f"\n Total píxeles del ROI extraídos: {len(pixel_values):,}")
    print(f"   Rango de intensidad: {np.min(pixel_values):.1f} -
{np.max(pixel_values):.1f}")
    print(f"   Promedio: {np.mean(pixel_values):.2f}\n")

    # Paso 3: Aplicar K-means
    print(f"🔄 Ejecutando K-means con {n_clusters} clusters...")
    inicio_kmeans = time.time()

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100,
0.2)
    _, labels, centers = cv2.kmeans(
        pixel_values, n_clusters, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS
    )

    tiempo_kmeans = time.time() - inicio_kmeans
    tiempo_total = time.time() - inicio_total

    centers = centers.flatten()
    labels = labels.flatten()

    # Ordenar centroides
    sorted_indices = np.argsort(centers)
    centers_sorted = centers[sorted_indices]

    print(f"✅ K-means completado en {tiempo_kmeans:.2f}s")
    print(f"   Tiempo total (incluyendo extracción ROI):
{tiempo_total:.2f}s\n")
```

```
    # Paso 4: Calcular estadísticas de cada cluster
    print(f"{'='*70}")
    print(f"{'Cluster':<10} | {'Centro':<12} | {'Mínimo':<10} |
{'Máximo':<10} | {'Píxeles':<12}")
    print("-" * 70)

    estadisticas = {}

    for i in range(n_clusters):
        cluster_pixels = pixel_values[labels == i]
        if cluster_pixels.size > 0:
            c_min = np.min(cluster_pixels)
            c_max = np.max(cluster_pixels)
            c_center = centers[i]
            c_count = len(cluster_pixels)

            estadisticas[i] = {
                'centro': c_center,
                'minimo': c_min,
                'maximo': c_max,
                'pixeles': c_count
            }

            print(f"{i:<10} | {c_center:<12.2f} | {c_min:<10.0f} |
{c_max:<10.0f} | {c_count:<12,}")

    print(f"{'='*70}\n")

    # Exportar centroides
    print("📋 CENTROIDES ORDENADOS (para usar en testset):")
    print(f"centers_roi = np.array({list(np.round(centers_sorted, 2))},
dtype=np.float32)\n")

    return centers_sorted, tiempo_total, estadisticas

# Ejecutar con 4 clusters
centers_roi, tiempo_roi, stats_roi = kmeans_en_roi(TRAIN_DIR, n_clusters=4)
```

Usamos los centroides calculados en el trainset (ROI) para segmentar nuevas imágenes del testset

```
def aplicar_clusters_roi_testset(image_path, centers_roi, visualizar=True):
    """
    Aplica los centroides del ROI a una nueva imagen del testset.
```

```
    Args:
        image_path: Ruta de la imagen de test
        centers_roi: Centroides calculados del trainset
        visualizar: Si True, muestra resultados

    Returns:
        labels_roi: Etiquetas de cluster para cada píxel del ROI
        mask: Máscara del ROI
    """
    # Cargar imagen
    img = cv2.imread(str(image_path), cv2.IMREAD_GRAYSCALE)
    if img is None:
        print("❌ No se pudo cargar la imagen")
        return None, None

    # Extraer ROI
    mask = tecnica_1_umbral_morfologia(img, visualizar=False)

    # Obtener píxeles del ROI
    roi_pixels = img[mask > 0].astype(np.float32)

    # Asignar a cluster más cercano
    distances = np.abs(roi_pixels[:, np.newaxis] - centers_roi)
    labels_roi = np.argmin(distances, axis=1)

    if visualizar:
        # Crear imagen segmentada
        segmented = np.zeros_like(img, dtype=np.uint8)
        roi_indices = np.where(mask > 0)

        for i in range(len(centers_roi)):
            cluster_mask = labels_roi == i
            segmented[roi_indices[0][cluster_mask],
roi_indices[1][cluster_mask]] = int(centers_roi[i])

        # Paleta de colores para visualización
        colors = [
            [128, 0, 128],   # Cluster 0: Púrpura (baja temperatura)
            [0, 0, 255],     # Cluster 1: Rojo (media-baja)
            [0, 165, 255],   # Cluster 2: Naranja (media-alta)
            [0, 255, 255]    # Cluster 3: Amarillo (alta temperatura)
        ]
```

```
        colored_img = np.zeros((img.shape[0], img.shape[1], 3),
dtype=np.uint8)
        for i in range(len(centers_roi)):
            cluster_mask = labels_roi == i
            colored_img[roi_indices[0][cluster_mask],
roi_indices[1][cluster_mask]] = colors[i]

        # Visualización
        plt.figure(figsize=(18, 5))

        plt.subplot(1, 4, 1)
        plt.imshow(img, cmap='gray')
        plt.title('Original')
        plt.axis('off')

        plt.subplot(1, 4, 2)
        plt.imshow(mask, cmap='gray')
        plt.title('Máscara ROI')
        plt.axis('off')

        plt.subplot(1, 4, 3)
        plt.imshow(cv2.bitwise_and(img, img, mask=mask), cmap='hot')
        plt.title('ROI Extraído')
        plt.colorbar(shrink=0.8)
        plt.axis('off')

        plt.subplot(1, 4, 4)
        plt.imshow(cv2.cvtColor(colored_img, cv2.COLOR_BGR2RGB))
        plt.title('Clusters en ROI')
        plt.axis('off')

        plt.tight_layout()
        plt.show()

        # Estadísticas
        print(f"\n📊 Estadísticas de la imagen:")
        print(f"   Píxeles totales: {img.shape[0] * img.shape[1]:,}")
        print(f"   Píxeles ROI: {len(roi_pixels):,}")
        print(f"   % ROI:
{len(roi_pixels)/(img.shape[0]*img.shape[1])*100:.2f}%\n")

        print(f"   Distribución de clusters:")
        for i in range(len(centers_roi)):
            count = np.sum(labels_roi == i)
```

```
            print(f"   Cluster {i} (centro={centers_roi[i]:.1f}): {count:,}
píxeles ({count/len(roi_pixels)*100:.1f}%)")

    return labels_roi, mask

# Probar con 3 imágenes del testset
test_images = list(TEST_DIR.glob('*.tif'))[:3]

print(f"\n{'='*70}")
print(f"APLICACIÓN AL TESTSET")
print(f"{'='*70}\n")

for idx, img_path in enumerate(test_images):
    print(f"\n[{idx+1}/{len(test_images)}] Procesando: {img_path.name}")
    print("-" * 70)
    labels, mask = aplicar_clusters_roi_testset(img_path, centers_roi,
visualizar=True)
```

Verificamos que el método se acopla a todas las imágenes del testset.

```
def validar_testset_completo(test_dir, centers_roi):
    """
    Aplica los centroides a todo el testset y genera estadísticas.
    """
    test_images = list(Path(test_dir).glob('*.tif'))

    print(f"\n{'='*70}")
    print(f"VALIDACIÓN EN TESTSET COMPLETO ({len(test_images)} imágenes)")
    print(f"{'='*70}\n")

    resultados = []
    errores = 0

    for idx, img_path in enumerate(test_images):
        try:
            # Cargar y procesar
            img = cv2.imread(str(img_path), cv2.IMREAD_GRAYSCALE)
            if img is None:
                errores += 1
                continue
```

```
            mask = tecnica_1_umbral_morfologia(img, visualizar=False)
            roi_pixels = img[mask > 0].astype(np.float32)

            if len(roi_pixels) == 0:
                errores += 1
                continue

            # Asignar clusters
            distances = np.abs(roi_pixels[:, np.newaxis] - centers_roi)
            labels = np.argmin(distances, axis=1)

            # Calcular distribución
            distribucion = [np.sum(labels == i) for i in
range(len(centers_roi))]

            resultados.append({
                'imagen': img_path.name,
                'roi_pixels': len(roi_pixels),
                'distribucion': distribucion
            })

            if (idx + 1) % 10 == 0:
                print(f"   [{idx+1}/{len(test_images)}] Procesadas...")

        except Exception as e:
            errores += 1
            continue

    print(f"\n✅ Procesamiento completado:")
    print(f"   Exitosas: {len(resultados)}/{len(test_images)}")
    print(f"   Errores: {errores}\n")

    # Estadísticas agregadas
    if len(resultados) > 0:
        distribucion_promedio = np.mean([r['distribucion'] for r in
resultados], axis=0)
        roi_promedio = np.mean([r['roi_pixels'] for r in resultados])

        print(f"{'='*70}")
        print(f"ESTADÍSTICAS AGREGADAS DEL TESTSET")
        print(f"{'='*70}")
        print(f"Píxeles ROI promedio: {roi_promedio:.0f}\n")
        print(f"Distribución promedio de clusters:")
        for i in range(len(centers_roi)):
```

```
            print(f"   Cluster {i} (centro={centers_roi[i]:.1f}):
{distribucion_promedio[i]:.0f} píxeles
({distribucion_promedio[i]/roi_promedio*100:.1f}%)")
        print(f"{'='*70}\n")

    return resultados

# Validar todo el testset
resultados_testset = validar_testset_completo(TEST_DIR, centers_roi)
```