

Replicación y Validación de Modelo Multitarea para Ultrasonido (FMC-UIA)

Juan Sebastian jaramillo Diaz

Cod:2212273

juan2212273@correo.uis.edu.co

Resumen—Este reporte documenta la replicación técnica y validación de una arquitectura de aprendizaje profundo basada en el repositorio *Foundation Model Challenge*. Debido al volumen del dataset original (40,000 imágenes), se implementó un pipeline de ingeniería de datos para asegurar la integridad referencial y reducir la muestra de forma estratificada al 15% (5,965 imágenes). El sistema utiliza un esquema de *Hard Parameter Sharing* con un codificador EfficientNet-B4 para resolver tareas de segmentación, clasificación, detección y regresión.

I. REDUCCIÓN DE LA DATA

Se desarrollaron scripts específicos para adaptar los datos al entorno de ejecución.

A. 2.1. Auditoría de Archivos

Se ejecutó un script de inspección para validar la correspondencia entre los archivos CSV (metadatos) y los archivos físicos.

- **Validación:** Se verificó la existencia de cada imagen listada en los CSV.
- **Regla de Segmentación:** Se aplicó un filtro estricto para las tareas de segmentación. Si la imagen original no contaba con su archivo de máscara binaria correspondiente en el disco, la entrada es eliminada, sin embargo cada imagen del dataset cuenta con su máscara correspondiente entonces no fue necesario aplicar esta estrategia.

B. Subconjunto Estratificado

Se generó un dataset reducido (*Train Mini*) compuesto por 5,965 imágenes. Se utilizó un muestreo estratificado basado en el `task_id`. Esto redujo el volumen de datos al 15% manteniendo la distribución porcentual de tareas idéntica a la fuente original.

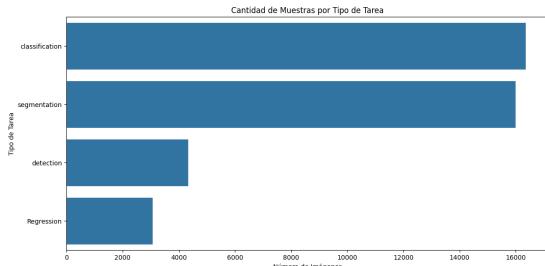


Fig. 1. Volumen de data del conjunto original

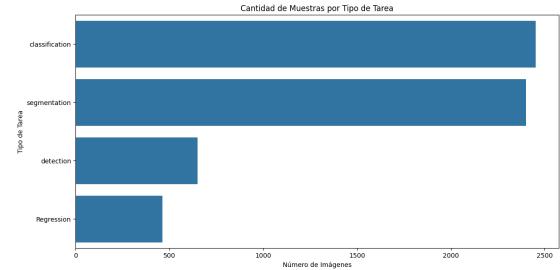


Fig. 2. Volumen de data del subconjunto

II. ARQUITECTURA DEL SISTEMA

El análisis del código fuente (`model_factory.py`) indica que el sistema implementa *Hard Parameter Sharing*.

- **Backbone:** Modelo **EfficientNet-B4** pre-entrenado en ImageNet. Funciona como extractor de características compartido.
- **Enrutamiento:** El flujo de datos se divide según la tarea:
 - **Clasificación y Regresión:** Usan *Pooling* global sobre las características del encoder.
 - **Segmentación y Detección:** Usan un decodificador **FPN (Feature Pyramid Network)** para recuperar resolución espacial.

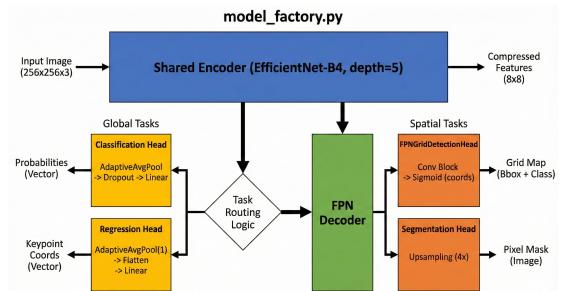


Fig. 3. FPN Encoder para Detección y segmentación

III. MANEJO DE DESBALANCE DE DATOS

El dataset presenta un desbalance numérico significativo: Clasificación (41%) y Segmentación (40%) frente a Detección (11%) y Regresión (8%).

El análisis del script de entrenamiento (`train.py`) determinó que este desbalance se gestiona mediante el componente

MultiTaskUniformSampler. Este muestreador fuerza al *DataLoader* a construir lotes (*batches*) que contienen ejemplos de una sola tarea a la vez. Esto garantiza que el modelo actualice sus pesos basándose en tareas minoritarias con la misma frecuencia que las mayoritarias, independientemente de la cantidad total de imágenes.

IV. RESULTADOS CON SET DE TRAIN

Métricas obtenidas en el conjunto de validación del subconjunto (15% de los 5,965 datos procesados):

A. Segmentación

El modelo demostró capacidad para delimitar estructuras anatómicas complejas.

Métrica	Valor Promedio
Dice Score (DSC)	0.7983
Hausdorff Distance (HD)	72.14

TABLE I

DESEMPEÑO GLOBAL EN LAS 12 TAREAS DE SEGMENTACIÓN.



Fig. 4. Resultados cualitativos de Segmentación. Máscaras predichas sobre ecografías.

B. Clasificación

Esta modalidad obtuvo el desempeño más robusto del sistema.

Métrica	Valor Promedio
Accuracy	90.80%
AUC	0.8735
F1-Score (Weighted)	0.9061

TABLE II

DESEMPEÑO GLOBAL EN LAS 9 TAREAS DE CLASIFICACIÓN.

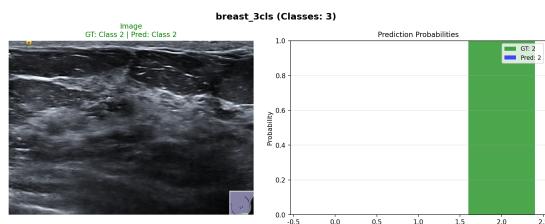


Fig. 5. Resultados de Clasificación. Etiquetas y confianza del modelo.

C. Detección

La localización de objetos mediante cajas delimitadoras presentó un reto mayor debido a la variabilidad de tamaño de las lesiones.

- **IoU Promedio:** 0.3982

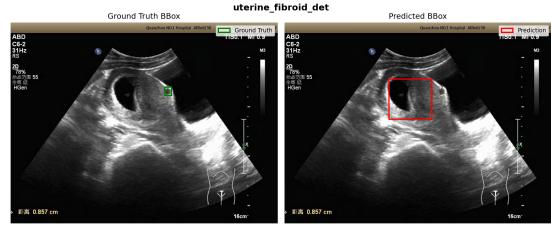


Fig. 6. Resultados de Detección. Cajas delimitadoras predichas.

A pesar de esto, en algunos casos se llegó a una detección que puede considerarse acertada, como se muestra en la Figura 7.

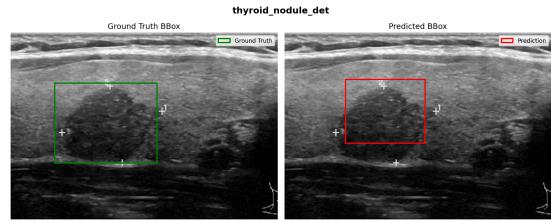


Fig. 7. Resultados de Detección. Cajas delimitadoras predichas.

D. Regresión

La estimación de puntos clave biométricos mostró convergencia en la tendencia general, aunque con margen de error en píxeles absolutos.

- **MRE:** 90.44

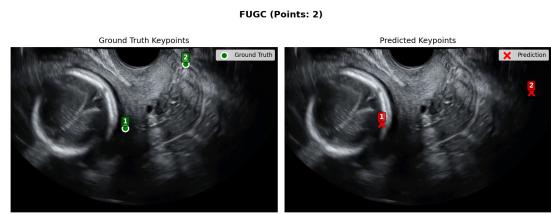


Fig. 8. Resultados de Regresión. Predicción de coordenadas (x, y).

listings xcolor

V. ESTRATEGIAS DE MEJORA IDENTIFICADAS

Con base en el análisis del código base y los resultados obtenidos (donde Detección y Regresión mostraron un rendimiento inferior), se proponen las siguientes modificaciones concretas al código para iteraciones futuras.

A. Refinamiento de Funciones de Pérdida

En *train.py*, las funciones de pérdida actuales son estándar. Para mejorar la regresión (sensible a outliers) y la detección (desbalanceada), se propone modificar el diccionario *loss_functions*:

```

1 # train.py - Modificación propuesta
2 loss_functions = {
3     'segmentation': smp_losses.DiceLoss(mode='
4         multiclass'),
5     'classification': nn.CrossEntropyLoss(),
6 }
```

```

5     # Cambio: De MSELoss a SmoothL1Loss para
6     # robustez ante outliers
7     'Regression': nn.SmoothL1Loss(beta=1.0),
8     # Cambio: Integracion futura de FocalLoss para
9     # Deteccion
10    'detection': FocalLoss(alpha=0.25, gamma=2.0)
11 }
```

Listing 1. Propuesta de cambio en train.py

```

6     loss.backward()
7
8
9 if (batch_idx + 1) % ACCUMULATION_STEPS == 0:
10    optimizer.step()
11    optimizer.zero_grad()
```

Listing 4. Logica de acumulacion en train.py

B. Aumentación Geométrica Específica

El pipeline actual aplica ruido gaussiano y cambios de brillo. Para ultrasonido, las deformaciones elásticas son críticas para simular la variabilidad de tejidos. Se propone actualizar `train_transforms` en `train.py`:

```

1 train_transforms = A.Compose([
2     A.Resize(256, 256),
3     # Nuevas transformaciones geometricas propuestas
4     :
5     A.ElasticTransform(alpha=120, sigma=120 * 0.05,
6     p=0.5),
7     A.GridDistortion(p=0.5),
8     A.HorizontalFlip(p=0.5), # Esencial para
9     # clasificacion
10    A.Normalize(...),
11    ToTensorV2(),
12 ], bbox_params=...)
```

Listing 2. Nuevas transformaciones en train.py

C. Rediseño del Cabezal de Regresión

La arquitectura actual en `model_factory.py` utiliza *Pooling* global, destruyendo la información espacial necesaria para localizar puntos clave. Se propone sustituir el `RegressionHead` por una estructura densa progresiva:

```

1 class RegressionHead(nn.Module):
2     def __init__(self, in_channels: int, num_points: int):
3         super().__init__()
4         # Propuesta: Mantener mas info espacial o
5         # aumentar capacidad
6         self.block = nn.Sequential(
7             nn.AdaptiveAvgPool2d(1),
8             nn.Flatten(),
9             nn.Linear(in_channels, 512),
10            nn.ReLU(),
11            nn.Dropout(0.3), # Regularizacion
12            nn.Linear(512, num_points * 2)
13        )
14
15    def forward(self, features: list):
16        return self.block(features[-1])
```

Listing 3. Mejora estructural en model_factory.py

D. Acumulación de Gradientes

Dado que el `BATCH_SIZE` está limitado por la memoria de la GPU. Se propone implementar acumulación de gradientes en el bucle de entrenamiento para simular lotes más grandes:

```

1 # train.py - Dentro del bucle for batch in loop:
2 ACCUMULATION_STEPS = 4 # Simula batch_size = 80
3
4 loss = loss_functions[task_name](final_outputs,
5     labels)
5 loss = loss / ACCUMULATION_STEPS # Normalizar loss
```