Draft version November 13, 2024
Typeset using LaTeX **twocolumn** style in AASTeX631

Reporte solución equipo Barrienlens Astrodatatón 2024

Felipe Urcelay,[1] Joaquin Hernandez,[1] and Rodrigo Ugarte[1]

[1] *Instituto de Astrofísica, Pontificia Universidad Católica de Chile*
Av. Vicuña Mackenna 4860, Macul
Región Metropolitana, 7820436, Chile

1. DATA PREPARATION

Para optimizar la carga y manipulación de datos, especialmente el límite I/O que establece Google Collab (la plataforma sobre la cual nuestro grupo trabajó) se agruparon los archivos .npy entregados en .HDF5 los cuales tienen la ventaja de que pueden ser cargados parcialmente en chunks de distintos tamaños.

1.1. *Data Augmentation*

Se aplican todas las transformaciones básicas tanto a los tensores de entrada (los mapas de elipticidad ϵ) como al mapa de salida (κ). Estas transformaciones se realizan de manera aleatoria aunque de la misma manera a las 3 matricez:

1. Rotaciones: Entre 0, 90, 180 y 270 grados
2. Flips: Se reflejan los mapas tanto vertical como horizontalmente

2. MODELO

Uno de los aspectos que mejor lograba englobar las 3 métricas utilizadas para puntuar los resultados (DPEAKS, DICEE Y WMAPE) es la necesidad de preservar la estructura espacial y las características locales de los datos de entrada. Dado que el objetivo principal es lograr una reconstrucción precisa de las distribuciones de masa, nuestra estrategia se enfocó en arquitecturas de redes neuronales convolucionales especializadas en la tarea de mapeo de imagen a imagen.

En el paper de referencia Hong et al. (2021) se utiliza una arquitectura ResNet (especialmente útiles para problemas de segmentación de imágenes). Inspirados en este tipos de arquitecturas decidimos utilizar una muy similar llamada UNet, que básicamente es un encoder-decoder (es decir, reduce secuencialmente mediante distintas capas la matriz de entrada hasta obtener un vector muy informativo pero con nula resolución, para luego ir aumentando la resolución mediante convoluciones transpuestas hasta recuperar la original) que utiliza skip-connections para preservar esta información espacial con un muy alto nivel de resolución, reduciendo el problema también a uno tipo segmentación en las que cada píxel debe ser clasificado correctamente (pix2pix).

2.1. *U-net*

Como mencionamos, la arquitectura de nuestra solución corresponde a una UNet, esta sigue la arquitectura clásica de encoder-decoder con skip connections, incorporando varias optimizaciones que mejoran la precisión, detalladas a continuación:

- **Capas de Entrada y Configuración de Canales**: La entrada del modelo es una imagen de tamaño $128 \times 128 \times 3$, correspondiente a las dos componentes de elipticidad y el error de medición. Opcionalmente, se puede agregar un cuarto canal si se incluye el mapa KS93, permitiendo al modelo aprender patrones adicionales en el mapeo hacia el mapa de masa proyectada (que finalmente terminamos descartando).
- **Capas de Downsampling (Encoder)**: El encoder está compuesto de capas de downsampling. Cada capa realiza las siguientes operaciones:
  - **Convolución**: Reduce la resolución de la entrada con un kernel de tamaño especificado, aplicado con un stride de 2 y padding `same`.
  - **Batch Normalization**: Aplicada en todas las capas excepto en la primera, para estabilizar el entrenamiento y mejorar la convergencia.
  - **Leaky ReLU**: Utilizada como función de activación para introducir no linealidad.

  En cada nivel de downsampling, el número de filtros se incrementa progresivamente, comenzando en 64 y alcanzando un máximo de 512.

riugarte@uc.cl

- **Capas de Upsampling (Decoder)**: El decoder utiliza capas de upsampling, que realiza las siguientes operaciones:
  - **Transposición**: Aumenta la resolución espacial para restaurar el tamaño de la imagen.
  - **Batch Normalization**: Normaliza las activaciones después de cada transposición convolucional.
  - **Dropout**: Utilizado en las tres primeras capas de upsampling para reducir el sobreajuste, con una probabilidad de droput de 0.5.
  - **ReLU**: Función de activación aplicada después de cada capa de upsampling para permitir la reconstrucción.

  Durante el upsampling, el número de filtros se reduce progresivamente en el orden inverso al encoder.

- **Manejo de Bordes con Reflection Padding y Cropping**: Para minimizar los efectos de borde (el problema de que muchos peaks 'cortados' en los bordes no se lograban reconstruir/detectar correctamente), se implementa una capa de *Reflection Padding* antes de la primera convolución. Esto utiliza el reflejo de los bordes de la imagen para mejorar la precisión en las áreas de borde. Después del paso por la red, una capa de *Cropping* elimina el padding adicional, restaurando el tamaño de la salida a 128×128.

- **Skip Connections**: Las *skip connections* como ya se explicó permiten que las activaciones de cada capa del encoder se pasen directamente a las capas correspondientes en el decoder mediante concatenación. Esto ayuda a preservar detalles finos y la estructura de la imagen de entrada.

- **Capa de Salida**: La capa final es una transposición convolucional con un kernel del mismo tamaño que las demás capas de upsampling, generando una salida de tamaño $128 \times 128 \times 1$, correspondiente al mapa de masa proyectada. Utilizamos una función de activación `tanh` en esta capa para limitar los valores generados al rango válido para la convergencia .

3. LOSS FUNCTIONS

En esta sección, nos enfocamos en diseñar funciones de pérdida (*loss functions*) personalizadas que ayuden al modelo a capturar tanto los valores globales como los detalles específicos de las imágenes, como los picos, bordes y estructuras importantes.

3.1. *MultiFocal Loss*

Nos basamos en la métrica *Focal Loss* de Hong et al. (2021), que corresponde a la diferencia cuadrática media entre los pixeles de la predicción y el objetivo, ponderados por la amplitud relativa al máximo global de la convergencia:

$$\mathcal{L} = w_f(x) \left[\kappa_{pred} - \kappa_{truth}\right]^2 \tag{1}$$

donde el peso focal $w_f(x)$ se define como:

$$w_f(x) = 1 + \frac{|\kappa_{truth}|}{max|\kappa_{truth}|} \tag{2}$$

Definimos la función de pérdida *Multifocal Loss* generalizando la definición del peso para incluir distintos componentes w_i, cada uno reforzando distintos componentes de la convergencia, ponderados por coeficientes a_i:

$$w(x) = \frac{a_i\, w_i(x)}{\sum_i a_i} \tag{3}$$

Para escoger los pesos adecuados realizamos un análisis preliminar entrenando el modelo UNet utilizando la diferencia cuadrática media (MSE) como función de pérdida. Estudiamos el resultado de la convergencia predicha por esta red en comparación con la convergencia esperada.

En base a este análisis diseñamos los siguientes pesos w_i (todos en el rango 0-1):

- **focal** (w_f): definido por la eq. 2, pondera los pixeles en base a su comparación con el máximo global.

- local (w_l): equivalente al peso w_f pero definido de manera local, ponderado por el máximo de convergencia en una vecindad de 8x8 pixeles. Este peso es más grande para los "clumps" de masa, independientemente de su magnitud global. Esto se fundamenta en que la señal de lentes débiles se basa en el shear, que no solo depende de la magnitud de la convergencia sino en su variación espacial. Una masa grande pero constante no produce shear, mientras que una masa concentrada produce una mayor señal. Esto favorece que la red detecte las estructuras a pequeña escala. Incluimos este peso solo para los pixeles en el 30% de mayor convergencia.

- **peaks** (w_p): Notamos que en cuanto a la métrica DPEAKS, la red logra generar un peak en la locación adecuada en casi todos los casos, sin embargo esta suele fallar en la reproducción de las amplitudes relativas entre estos peaks, por lo que

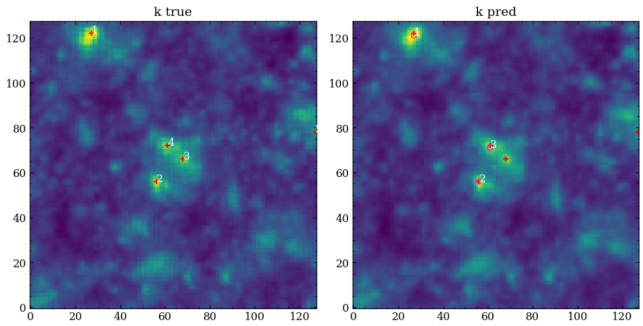

Figure 1. Comparación entre los peaks más prominentes en la convergencia objetivo (κ true) y la predicha por el modelo (κ pred). En el primer caso se indican los 5 peaks más intensos con cruces. En el segundo caso se muestran los 3 más intensos señalados con círculos, sobre los 5 más intensos de κ true. Puede verse que el modelo genera un peak en la posición 3 de κ true pero con una intensidad menor al 4to peak.

puede confundir el orden de estos. Esto puede apreciarse en la figura 1 en que el modelo predice correctamente los peaks pero confunde las amplitudes del 3er peak con el 4to. Por este motivo consideramos el peso w_p, que pondera en mayor magnitud los pixeles al rededor de los 5 peaks más intensos en κ_{truth}, favoreciendo que la red prediga correctamente no solo la amplitud de los tres primeros, sino que también los dos siguientes con los que suele confundirse.

- **structure** (w_s): Al analizar el resultado de la métrica DICEE, apreciamos que la red logra predecir correctamente casi todos los pixeles que se conforman la estructura a gran escala, tanto los pixeles que se encuentran por sobre la media (estructuras), como los que se encuentran bajo esta (vacíos). Sin embargo tiene un mayor error en el borde de estas estructuras, es decir, los pixeles que se encuentran cerca de la media. Esto puede apreciarse en la figura 2 en que la red solo falla en unos pocos pixeles al rededor de las estructuras. Por este motivo w_s le da un peso extra a los pixeles al rededor de la media de $\kappa_{truth} \pm 0.001$.

- **edges** (w_e): Notamos que suele haber un mayor error de predicción en los bordes del mapa, por lo que este peso le da un valor adicional a un marco de 5 pixeles al borde de este.

El resultado de los componentes de MultiFocal Loss se muestra en la figura 3.

3.2. *Pérdida Combinada*

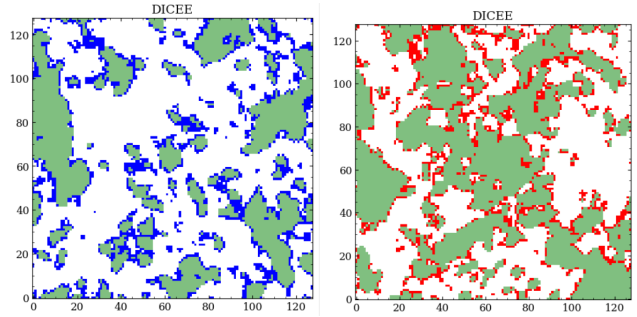

Figure 2. Evaluación de la métrica DICEE para dos ejemplos de predicciones del modelo. En verde se encuentran los verdaderos positivos (estructuras), en blanco los verdaderos negativos (no estructuras), en azul los falsos positivos, y en rojo los falsos negativos. Se puede apreciar que en el primer ejemplo se sobre estima las estructuras y en el segundo se subestima.

Además de la métrica anterior, diseñamos la métrica *Combined Loss* que integra MultiFocal Loss junto a las funciones WMAPE, DICEE y DPEAKS, asignando un peso específico a cada una según cuál quisiéramos minimizar en mayor medida. Utilizamos esta métrica para el fine-tuning de la red tras el entrenamiento.

4. BÚSQUEDA DE HYPERPARÁMETROS

Para la búsqueda de hiperparámetros en nuestra arquitectura U-Net, empleamos una metodología organizada para optimizar aquellos parámetros que impactan directamente en el rendimiento del modelo.

1. Selección de Hiperparámetros Clave Primero identificamos cuáles eran los hiperparámetros más importantes para nuestra tarea. Estos incluían el tamaño de los filtros en las capas convolucionales, la profundidad de la red (número de capas), la extensión de bordes (para mejorar el rendimiento en los límites de las imágenes), el learning rate y batch size para controlar la tasa de aprendizaje y optimizar la convergencia. La selección de estos parámetros se basó en la comprensión de cómo afectan el aprendizaje y la capacidad de generalización de la U-Net.

2. Estrategia de Optimización de Hiperparámetros Para explorar el espacio de posibles configuraciones de manera eficiente, utilizamos Keras Tuner O'Malley et al. (2019), una herramienta que automatiza la búsqueda de hiperparámetros. En lugar de probar cada combinación posible (lo que sería muy costoso en términos de tiempo y recursos), aplicamos Bayesian Optimization. Esta técnica explora de manera estratégica el espacio de hiperparámetros, identificando patrones en las iteraciones previas para seleccionar solo aquellos valores que tienen más probabilidades de mejorar el rendimiento del modelo.

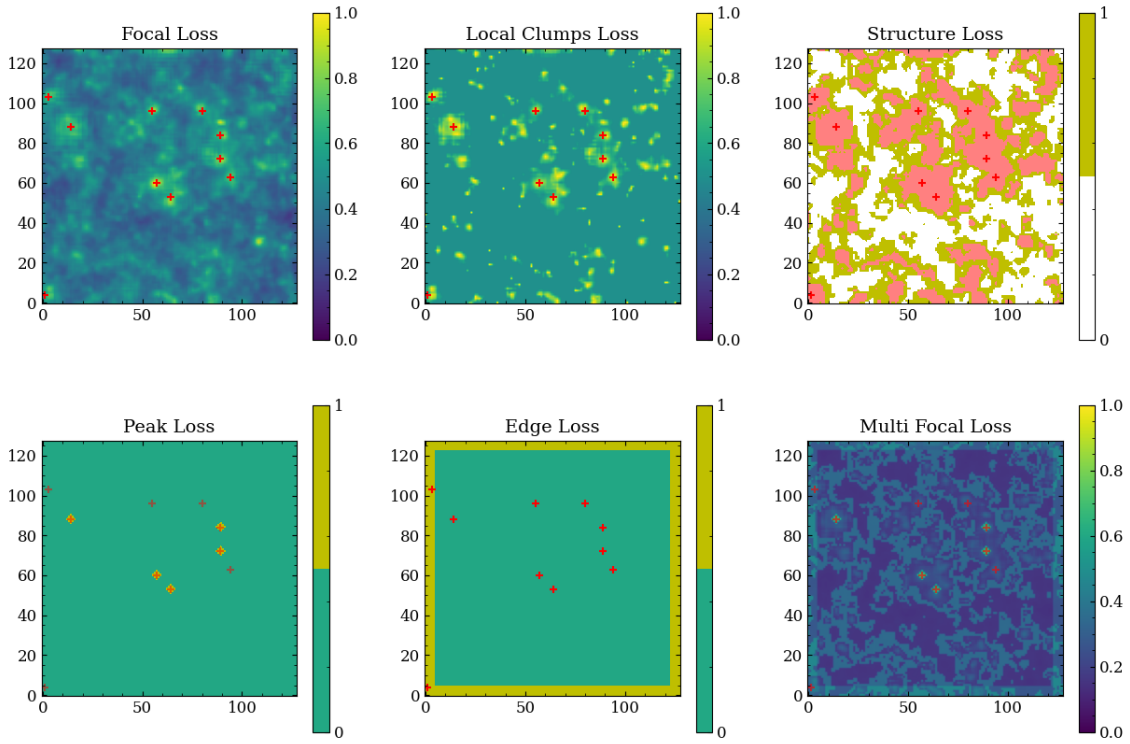

Figure 3. Se muestran los 5 pesos diseñados para la métrica MultiFocal Loss para un ejemplo aleatorio. En todos ellos se indican lso 10 peaks más intensos con cruces rojas. En el caso del peso Structure Loss, en rosado se indican las estructuras para una mejor visualización, pero los pesos se indican en amarillo. El último panel muestra el mapa de pesos combinado usando los coeficientes a_i determinados en la búsqueda de hiperparámetros.

3. Evaluación de las Configuraciones de Hiperparámetros Para cada combinación de hiperparámetros, entrenamos el modelo por 30 épocas utilizando 10 000 ejemplos de entrenamiento y evaluamos su rendimiento en un conjunto de validación usando las métricas WMAPE, DICEE y DPEAKS. También usamos una técnica llamada early stopping para detener el entrenamiento si el modelo dejaba de mejorar en el conjunto de validación, evitando así el sobreajuste y reduciendo el tiempo de entrenamiento.

4. Selección de la Configuración Óptima Después de completar la búsqueda, seleccionamos la configuración de hiperparámetros que proporcionó el mejor rendimiento en la métrica de validación. Esta configuración fue luego afinada (fine-tuning) en entrenamientos posteriores para asegurarnos de que la U-Net generalizara bien y obtuviera resultados consistentes en la tarea de predicción.

El resultado de esta búsqueda de hiperparámetros fue un modelo de 8 capas de convolución y 8 capas de convolución transpuesta, incluyendo *padding* reflectivo en la entrada y *cropping* en la salida. El learning rate óptimo fue de $1 \cdot 10^{-4}$ y el tamaño del *batch* 8. En cuanto al tamaño del *kernel*, durante la exploración probamos números pares de 2 a 10, siendo el óptimo de 2 pixeles. Sin embargo, posteriormente determinamos que un tamaño de 3 generaba un mejor rendimiento, ya que al ser igual que el tamaño del *stride* un kernel de 2 genera discontinuidades en la salida.

5. También realizamos búsqueda de hyperparámetros para nuestra MultiFocalLoss para ver según qué coeficientes a_i la red lograba minimizar las métricas que más nos interesaban. Se obtuvieron los siguientes pesos para esta función de pérdida:

- focal: 1.0
- local: 2.0
- peak: 3.0
- structure: 1.5
- edges: 1.0

Además, en esta búsqueda se corroboró que esta métrica supera a FocalLoss y MSE.

4.1. *Entrenamiento*

Se entrenó la red con los hiperparámetros óptimos y utilizando el total de datos de entrenamiento durante 50 épocas con la métrica MultifocalLoss, tras lo cual no

encontramos mejoras en la función de pérdida o en las tres métricas.

El resultado de esta red en el set de validación de esta red es:

- WMAPE: 1.53
- DICEE: 0.045
- DPEAKS: 20.3

Se usaron aproximadamente 76.000 ejemplos (es decir, un .tar completo) para validar la red durante todo el entrenamiento (las 50 epochs).

4.2. *Fine-Tuning*

Una vez el modelo alcanzó este plateau, realizamos un Fine-Tuning del modelo congelando los pesos de toda la red a excepción de la última capa. Durante el Fine-Tuning variamos la función de pérdida utilizando *CombinedLoss* para mejorar el rendimiento en una métrica específica.

Asignando un peso de 0.05 a WMAPE y 0.95 a MultiFocal Loss logramos mejorar la métrica WMAPE a costa de un leve aumento en las otras dos métricas. El modelo publicado en la entrega final obtuvo los siguientes resultados en el set de validación:

- WMAPE: 1.35
- DICEE: 0.051
- DPEAKS: 23.4

5. OTROS MODELOS

5.1. *Unet 2k*

Un modelo alternativo al modelo propuesto corresponde a UNet-2k que presenta dos kernels de distinto tamaño, uno de 3x3 que captura el detalle, y otro de 9x9 que permite analisis a mayor escala. En un entrenamiento de 20 épocas logramos un mejor rendimiento con este modelo que con el modelo anterior en la misma época, sin embargo debido a limitaciones de tiempo y recursos no logramos continuar con el entrenamiento de esta red.

5.2. *KS93*

Otro modelo que experimentamos utiliza una entrada adicional correspondiente a un mapa de convergencia proveniente del método KS93, que puede servir de referencia para el modelo. Este se añadía aleatoriamente a un 50% de los ejemplos para que no dependa únicamente de esta.

Se crearon mapas de KS93 a partir de aplicar una transformada de Fourier discreta a los mapas de elipticidad, con el fin de que el modelo lograra aprender como llegar de un solo input (ks93 al output (κ) pensando que lograría generalizar de mejor manera, pero resultó que nos empeoraban levemente las métricas. Estos se estimaron utilizando la expresión tal obtenida de 5.62 Introduction to Gravitational Lensing Lecture scripts (Massimo Meneghetti)

5.3. *Propuestas futuras*

Creemos que nuestro modelo puede beneficiarse ampliamente de capas de atención, que en conjunto con la función de pérdida MultiFocal Loss, permitirían optimizar los recursos de la red al aprender a focalizarse en los elementos importantes para mejorar las métricas buscadas.

REFERENCES

Hong, S. E., Park, S., Jee, M. J., Bak, D., & Cha, S. 2021, The Astrophysical Journal, 923, 266

O'Malley, T., Bursztein, E., Long, J., et al. 2019, KerasTuner, https://github.com/keras-team/keras-tuner

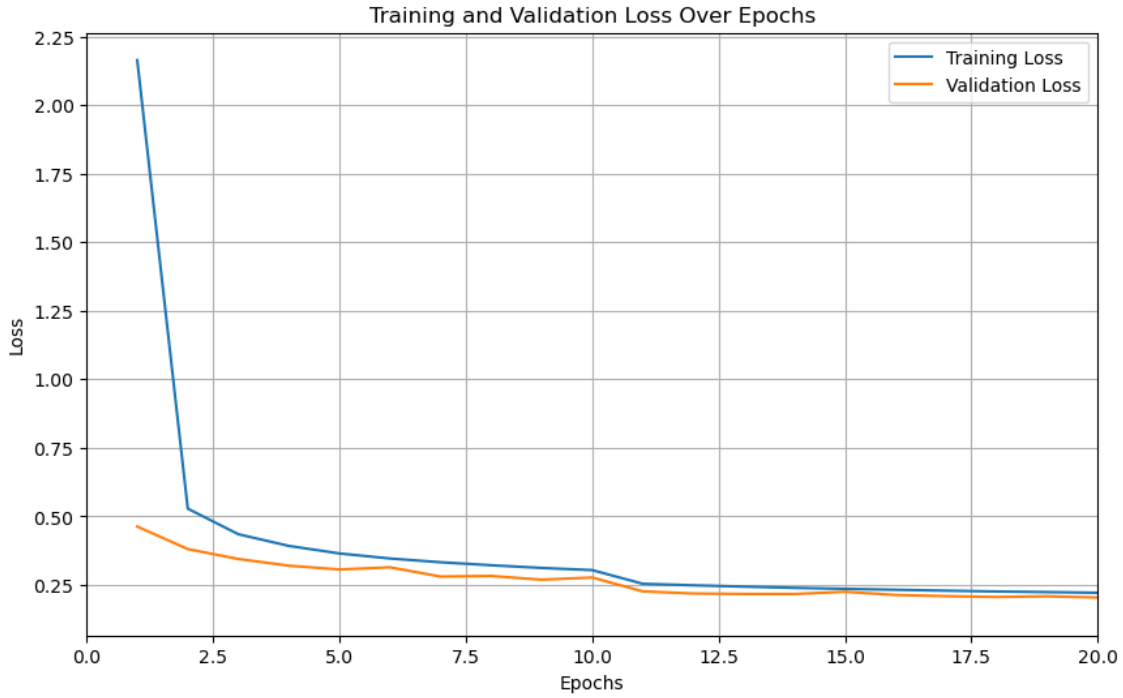

Figure 4. Se muestran las 20 primeras epochs de entrenamiento, donde se observa una clara convergencia de la loss para los datos de validación

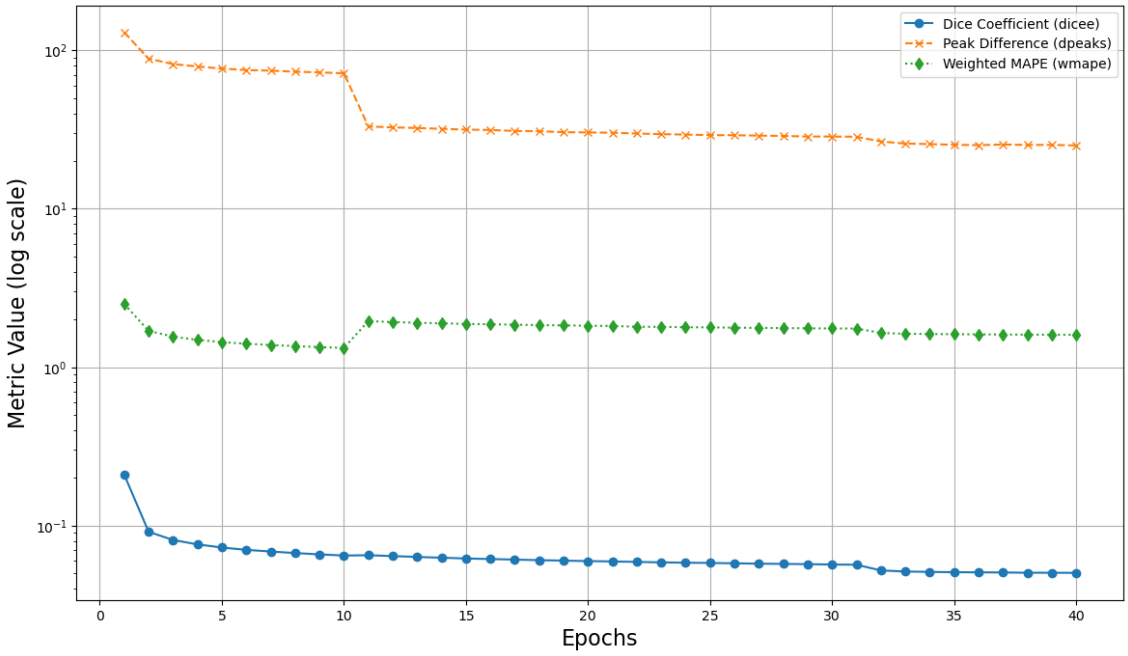

Figure 5. Se muestra la evolución de las métricas utilizadas para estimar la robustez del modelo durante el entrenamiento, el eje y se encuentra en escala logaritmica