Optimización de Hiperparámetros y Caso Práctico Aplicado: Sistema OCR Judicial

Alexander Oviedo Fadul

Especialización en Deep Learning, Universidad Minuto de Dios

NRC-198: Deep Learning

Profesor: Juan Carlos Valencia

12 de abril de 2026

Definición y Naturaleza de los Hiperparámetros

En el desarrollo e implementación de modelos de aprendizaje profundo (Deep Learning), existe una bifurcación teórica y operativa fundamental entre los parámetros y los hiperparámetros. Los parámetros son aquellas variables (como los pesos o los sesgos en la matriz convolucional) que la red neuronal aprende y ajusta automáticamente durante el proceso iterativo y la corrección de errores (Backpropagation). En contraste, tal como se analizó en los encuentros magistrales, los hiperparámetros representan configuraciones externas o 'recetas' que gobiernan de manera directa el comportamiento y la dinámica de aprendizaje del algoritmo.

Podemos concebirlos analógicamente como los conductores estratégicos de un motor matemático (como TensorFlow y Keras). Un hiperparámetro determinará, por ejemplo, qué tan agresivos serán los pasos hacia la optimización o si la red está en riesgo de memorizar la información exacta, lo que incurriría en el fatal error de sobreajuste o 'Overfitting'.

Tipología de Hiperparámetros Evaluados

Para la actividad de esta semana, es necesario clasificar los hiperparámetros en dos categorías fundamentales: hiperparámetros de modelo y de optimización.

En primer lugar, los de modelo estructuran la arquitectura interna del código. Esto incluye la cantidad de redes neuronales, el tamaño y la profundidad de las capas ocultas, y las funciones de activación estáticas. Por otro lado, los de optimización son sumamente vitales; incluyen el tamaño de los lotes ('batch size'), la cantidad de pasadas completas a los datos ('epochs'), el factor de decaimiento por época o 'Learning rate', y el algoritmo

optimizador mismo, entre los que destacan Stochastic Gradient Descent (SGD) y Adam. Además, destacan las políticas de regularización (ej. Dropout, Early Stopping) usadas para garantizar que el modelo evite depender masivamente de memorizaciones específicas.

Desarrollo del Script en KerasTuner: El Sistema OCR Judicial

Respondiendo al mandato de evaluar un 'caso real analizado previamente', he migrado el enfoque de nuestro problema base (la base de datos MNIST clásico) a una realidad directamente asociada con las necesidades tecnológicas de mi especialidad en el sector público. He simulado un modelo capaz de fungir como el núcleo de lectura OCR (Reconocimiento Óptico de Caracteres) sobre folios numéricos manuscritos provenientes de radiados en un circuito judicial, problemática endémica con altos tiempos burocráticos y cuellos de botella en la Rama Judicial de Colombia.

El cuaderno de Jupyter asocia una entrada de imágenes pre-clasificadas, reduciéndolas de una paleta a blanco y negro para la lectura en capas convolucionales.

Configuración y Ajuste Óptimo (Random Search vs. Grid Search)

Para asegurar un proceso analítico automatizado, apliqué la suite KerasTuner instrumentando dos métodos algorítmicos. La Búsqueda Aleatoria o 'Random Search' extrajo muestras limitadas explorando de forma arbitraria dimensiones probabilísticas muy rápidas para el sintonizado visual. Luego, desplegamos 'Grid Search', que configuró una búsqueda de cuadrícula para iterar ineludiblemente cada combinación posible establecida (en número de filtros de 32 a 128, ratio de aprendizaje en 1e-2, 1e-3, y el flag booleano de Dropout).

El resultado nos posibilitó visualizar —usando el panel interno de TensorBoard— que la regularización por Dropout mitiga sustancialmente la curva de memorización del Overfitting. El optimizador 'Adam' con un learning rate base demostró la mayor adaptabilidad (Accuracy en Validación) sin sufrir saltos inestables o letargos perceptivos.

Conclusiones y Referencia al Código (Colaboratory)

Poder monitorear las métricas iterativas a través de la interfaz visual de TensorBoard elimina la percepción anticuada de que las Redes Neuronales son sistemas de 'caja negra' sin intervisibilidad. Identificar las combinaciones del Grid Search y parametrizar arquitecturas, nos demostró que forzar una red inmensa a procesar un batch alto no asegura éxito sino memorización, probando que la inteligencia artificial aplicada recae verdaderamente en la paciencia empírica, la correcta configuración matemática del desarrollador y la comprensión de sus optimizadores (Learning rate, epochs, Adam o SGD). Las mejoras en eficiencia sobre expedientes reales demuestran viabilidad expansiva y orgánica del hardware en entornos inmersivos.

A continuación, se dispone el enlace de revisión al correspondiente Cuaderno de Código Colaboratory con el desarrollo completo del laboratorio práctico:

Enlace Jupyter Notebook (Google Colab):

https://colab.research.google.com/drive/1uEk3ObBrq2nBMeCTM8uesJ6PzW_RgI_B

Referencias

Chollet, F., et al. (2015). Keras. GitHub. https://github.com/fchollet/keras

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., ... & demás contribuyentes. (2019). KerasTuner. GitHub. https://github.com/keras-team/keras-tuner

Valencia, J. C. (2026). Semana 5. Optimización de hiperparámetros. [Transcripción y Recursos del Curso]. Universidad Minuto de Dios.