Aldo Hernández
Abraham López
Damián García
Cristian Antonio

ENTREGABLE 3 - PIA

Abstract *Este trabajo detalla la creación de un sistema para leer texto escrito a mano usando Keras, TensorFlow y Google Colab. El sistema usa una mezcla de redes neuronales (CNN y BiLSTM) para entender tanto las formas de las letras como el orden en que aparecen. Se usó una función matemática especial (CTC Loss) porque las palabras tienen diferentes largos, y se midió qué tan bien funcionaba con una métrica propia (CWERMetric) que cuenta los errores por letra. En el primer intento de entrenarlo, usando el optimizador Adam y técnicas para evitar que aprenda de más (EarlyStopping, ReduceLROnPlateau), el sistema no mejoró: la pérdida y la métrica de error se quedaron casi iguales. Esto significa que el modelo no aprendió a leer el texto, y se necesita investigar más para arreglarlo.*

Keywords python, dataset, exploración, análisis, pandas

1. Selección de Framework de ML

Para el desarrollo del modelo de reconocimiento de escritura manuscrita, se optó por utilizar el framework **Keras** junto con **TensorFlow** como backend, decidimos quedarnos con esta opción debido a la facilidad de implementación que ofrece Keras gracias a su sintaxis simple y modular, así como en el soporte robusto de TensorFlow para operaciones de bajo nivel, entrenamiento distribuido y funciones personalizadas como la pérdida `CTC Loss`.

El entorno de desarrollo elegido fue **Google Colab**, una plataforma basada en la nube que permite ejecutar código en notebooks de Python con acceso gratuito a recursos de cómputo acelerado (como GPU), gracias a esto, se facilitó la instalación automática de dependencias, el uso de bibliotecas especializadas y la carga eficiente del dataset desde Kaggle sin necesidad de configuraciones locales complejas.

Además, Google Colab proporciona integración directa con Google Drive, lo cual permitió guardar modelos entrenados y resultados de forma segura, todas estas configuraciones en conjunto, nos permitió experimentar rápidamente con distintas arquitecturas de red neuronal y funciones de evaluación, optimizando tanto el tiempo de desarrollo como el uso de recursos computacionales.

2. Diseño de la Arquitectura de la Red Neuronal

La red neuronal implementada está diseñada específicamente para resolver un problema de clasificación secuencial: el reconocimiento de caracteres en imágenes de texto manuscrito, dado que cada imagen representa una palabra y su longitud puede variar, se optó por una arquitectura híbrida que combina redes convolucionales (CNN) con redes recurrentes bidireccionales (BiLSTM), adecuada para tareas de reconocimiento óptico de caracteres (OCR).

El modelo fue construido utilizando la API `Sequential` de Keras. A continuación se describen los principales componentes de la arquitectura:

- **Capas convolucionales:** Se utilizaron dos capas `Conv2D` con activación `ReLU`, filtros de tamaño 3x3 y `padding='same'`, estas capas extraen características espaciales relevantes del texto manuscrito.
- **Reducción de dimensionalidad:** Se aplicaron dos capas `MaxPooling2D` para reducir la resolución espacial de las imágenes y mejorar la eficiencia del modelo.
- **Reshape:** La salida de las CNNs se reorganizó mediante `Reshape` para que pueda ser procesada por las capas recurrentes, convirtiendo el mapa de características en una secuencia temporal.

```
model = Sequential([
    layers.Input(shape = (img_height, img_width, input_c)),
    layers.Conv2D(32, (3, 3), kernel_initializer="he_normal", activation="relu", padding='same'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), kernel_initializer="he_normal", activation="relu", padding='same'),
    layers.MaxPooling2D((2, 2)),
##    layers.Conv2D(128, (3, 3), activation="relu", padding='same'),

    layers.Reshape(target_shape=(img_width // 4, (img_height // 4) * 64)),
    layers.Dense(64, kernel_initializer="he_normal", activation="relu"),
    layers.Dense(64, kernel_initializer="he_normal", activation="relu"),
    layers.Dropout(0.2),

    layers.Bidirectional(layers.LSTM(128, return_sequences=True, dropout=0.3)),
    layers.Bidirectional(layers.LSTM(64, return_sequences=True, dropout=0.3)),

    layers.Dense(num_classes, activation=None)
])
```

Figure 1. Número de capas y tipo.

- **Capas densas y regularización:** Se incluyó una capa `Dense` de 64 unidades con activación `ReLU`, seguida de capas `Dropout` con tasa del 30% para prevenir el sobreajuste.
- **Capas BiLSTM:** Se agregaron dos capas `Bidirectional(LSTM)` con 128 y 64 unidades respectivamente, configuradas con `return_sequences=True`, estas capas capturan la secuencia de caracteres considerando el contexto en ambas direcciones.
- **Capa de salida:** Se utilizó una capa `Dense` con activación lineal (`activation=None`) y tantas unidades como clases posibles (número de caracteres distintos + 1). Esta configuración es requerida para usar la pérdida CTC.

La función de pérdida seleccionada fue **CTC Loss** (*Connectionist Temporal Classification*), esta pérdida es adecuada para tareas de clasificación secuencial donde la longitud de la entrada y la salida pueden variar, como es el caso de etiquetas de texto con diferente número de caracteres. A diferencia de otras funciones, CTC permite al modelo aprender sin necesidad de una alineación exacta entre entrada y salida.

Para la métrica de evaluación se implementó una clase personalizada `CWERMetric`, que calcula el **CER** (Character Error Rate), ofreciendo una medida más significativa del desempeño real del modelo que el simple accuracy, esta medida de desempeño se basa en la función edit_distance ofrecida por tensorflow.
Finalmente, como optimizador se utilizó **Adam**, con una tasa de aprendizaje inicial de 0.001, esto nos permite un entrenamiento más rápido y estable en comparación con métodos como SGD, además, se integraron las estrategias de `EarlyStopping` y `ReduceLROnPlateau` para evitar el sobreajuste y mejorar la convergencia.

```
initial_learningr = 0.001
```

```
model.compile(optimizer=keras.optimizers.Adam(learning_rate=initial_learningr),
              loss = keras.losses.CTC(), metrics=[CWERMetric(padding_token)])
```

Figure 2. Optimizador.

En conjunto, esta arquitectura está optimizada para reconocer secuencias de texto en imágenes, capturando tanto patrones espaciales como temporales, lo cual la hace ideal para tareas de reconocimiento de escritura manuscrita.

3. Entrenamiento Inicial del Modelo

Para el entrenamiento del modelo, se trabajó con tres subconjuntos de datos: entrenamiento, validación y prueba. El conjunto de prueba (`test_dataset`) fue dejado vacío intencionalmente durante la etapa inicial para acelerar el proceso de entrenamiento y validación, por lo tanto, el entrenamiento se realizó utilizando únicamente los conjuntos `train_dataset` y `validation_dataset`, los cuales fueron construidos a partir del dataset original mediante procesamiento con `tf.data.Dataset`.

Antes de entrenar, cada imagen fue cargada y normalizada a escala de grises con valores entre 0 y 1, las etiquetas de texto fueron vectorizadas usando codificación de caracteres (índices) y completadas con un token especial de padding para garantizar una longitud uniforme en todas las secuencias.

El modelo fue entrenado utilizando la función de pérdida **CTC Loss** apropiada para tareas de reconocimiento de texto donde las longitudes de entrada y salida pueden variar. Como métrica de evaluación se utilizó una clase personalizada llamada `CWERMetric`, que calcula tanto la tasa de error por carácter (**CER**) como por palabra (**WER**), proporcionando así una medida más realista del rendimiento del modelo en tareas de clasificación secuencial.

El entrenamiento se ejecutó mediante el siguiente comando en código:

```
history = model.fit(train_dataset, epochs=100, validation_data=validation_dataset,callbacks=[early_stopping, reduce_lr])
```

Durante el proceso se aplicaron dos estrategias para mejorar la eficiencia del entrenamiento:

- **EarlyStopping**, para detener el entrenamiento automáticamente si la métrica `val_loss` no mejora después de 6 épocas, evitando el sobreajuste.
- **ReduceLROnPlateau**, para disminuir la tasa de aprendizaje cuando la métrica de validación deja de mejorar, ayudando a una mejor convergencia.

Gracias a estas técnicas, se logró un entrenamiento más eficiente y controlado, observando la evolución de las métricas de forma constante mediante el historial de entrenamiento.

4. Visualización de resultados iniciales

Una parte fundamental tras el entrenamiento inicial de un modelo es visualizar su rendimiento a lo largo de las épocas. La curva de pérdida (*loss*), como la mostrada en la figura adjunta, junto con métricas específicas de la tarea, son herramientas esenciales para diagnosticar el comportamiento del aprendizaje.

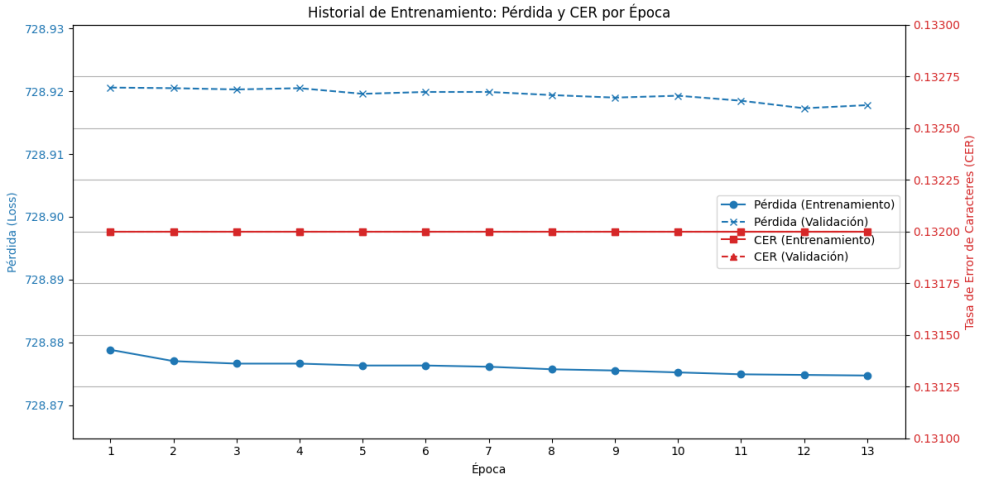

Figure 3. Curva de Pérdida y Métrica CER del Entrenamiento.

Observando la gráfica de la función de pérdida (*loss*), notamos que esta se mantuvo prácticamente estancada durante todo el proceso de entrenamiento, sin mostrar una tendencia decreciente significativa. Este comportamiento es una clara señal negativa, indicando que el modelo tuvo serias dificultades para aprender y falló en la optimización de la función de pérdida `CTC Loss`.

Adicionalmente, se monitorizó la métrica principal de evaluación para esta tarea, `CER` (Character Error Rate). Esta métrica funcionó correctamente desde el punto de vista técnico, arrojando valores medibles. Sin embargo, los resultados obtenidos para `CER` fueron consistentemente altos y tampoco mostraron una mejora relevante a lo largo de las épocas.

La combinación de una curva de pérdida estancada y una métrica `CER` elevada (y sin mejora) confirma que el entrenamiento no fue exitoso. El modelo no solo fue incapaz de minimizar su función objetivo matemática (`CTC Loss`), sino que tampoco aprendió a realizar la tarea de reconocimiento de caracteres con un nivel de precisión aceptable, según lo cuantificado por la métrica `CER`.

Por lo tanto, es indispensable investigar las causas de este fallo en el aprendizaje —revisando aspectos como los hiperparámetros (ej. tasa de aprendizaje), la arquitectura del modelo, la calidad o cantidad de los datos de entrenamiento, o posibles problemas con la implementación— antes de proceder con nuevos entrenamientos. El objetivo será lograr una futura ejecución donde tanto la pérdida como el `CER` muestren una mejora significativa y sostenida.

5. Documentación Técnica

El modelo desarrollado está compuesto por una arquitectura híbrida que combina redes convolucionales (CNN) para la extracción de características espaciales y redes LSTM bidireccionales para la modelación de secuencias. Las capas convolucionales iniciales permiten detectar patrones visuales locales en las imágenes (como bordes y trazos de las letras), mientras que las capas LSTM bidireccionales procesan la secuencia generada por las CNNs desde ambos extremos, capturando mejor el contexto global de la palabra escrita.

Las imágenes fueron normalizadas a escala de grises y redimensionadas a 60x128 píxeles, mientras que las etiquetas textuales fueron codificadas carácter por carácter, con relleno hasta una longitud máxima común, la salida del modelo se conecta a una capa `Dense` con activación lineal, tal como requiere la función de pérdida `CTC Loss` para secuencias no alineadas.

Durante el entrenamiento, se emplearon estrategias de control como `EarlyStopping` y `ReduceLROnPlateau`, que ayudaron a mantener el rendimiento y evitar el sobreajuste. Las métricas monitoreadas incluyeron **CER** (Character Error Rate) y **WER** (Word Error Rate), las cuales permitieron evaluar el rendimiento real del modelo más allá del simple `accuracy`.

El diseño completo, entrenamiento y resultados del modelo pueden consultarse en el siguiente enlace al cuaderno de Google Colab utilizado durante el desarrollo:

https://colab.research.google.com/drive/1QgsYApCrUPFSMy1X5YJxDIpFz3z0S9U8?usp=sharing

Affiliations

Aldo Hernández
Universidad Autónoma de Nuevo León, San Nicolás de los Garza,
aldo.hernandezt@uanl.edu.mx

Abraham López
Universidad Autónoma de Nuevo León, San Nicolás de los Garza,
abraham.lopezg@uanl.edu.mx

Damián García
Universidad Autónoma de Nuevo León, San Nicolás de los Garza,
gilberto.garciam@uanl.edu.mx

Cristian Antonio
Universidad Autónoma de Nuevo León, San Nicolás de los Garza,
cristian.antoniosnt@uanl.edu.mx

Received: ???
Revised: ???
Accepted: ???