Tecnológico de Monterrey

Reporte Final:

Modelo de ML para reconocimiento de CAPTCHAS

Presenta:

Castillo Sánchez Alan Rodrigo A01708668

2 jun 2025

Profesor:

Benjamín Valdés Aguirre

Desarrollo de aplicaciones avanzadas de ciencias computacionales

Módulo 2 - Inteligencia Artificial

Índice

Índice **1**
Introducción **2**
Dataset **3**
Descripción 3
Técnicas de escalamiento y aumentación 3
Análisis exploratorio 4
Balanceo de caracteres 5
Modelo CNN **6**
Resultados **8**
Primer modelo 8
Segundo modelo 11
Tercer modelo 13
Referencias **15**

Introducción

Hoy en día la mayoría de sitios web y aplicaciones, buscan prevenir accesos que potencialmente puedan causar un problema de acceso sus servicios o simplemente para evitar cualquier tipo de acción maliciosa en su contra, para este tipo de problemas es que se utilizan las pruebas tipo CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart), que ayudan, o ayudaban, a garantizar que era efectivamente una persona la que se estaba conectando a tus servicios y no una "máquina" o bots. Estas pruebas mundialmente conocidas, consisten básicamente en imágenes con una cantidad variable de caracteres alfanuméricos, las cuales tienen una multitud de variabilidad en cuanto a la imagen en sí: líneas cruzadas, distorsiones, baja resolución, "ruido" visual como puntos en patrones o cambios de colores contrastantes, deformación de los caracteres, entre muchas otras. Es por esto que se vuelve importante validar la veracidad o efectividad de estos tests, es decir que ciertamente sean capaces de evitar que una máquina pueda reconocer los caracteres correctamente, para verificar la seguridad que pueden darle a la web ante esto. Múltiples modelos de Deep Learning ya son capaces de realizar esta tarea hoy en día efectivamente, con una efectividad de hasta un 99.8% [1][2] sin embargo suelen ser modelos altamente "costosos" computacionalmente y que pueden ser complicados de replicar e implementar.

En este reporte se estará describiendo, analizando, revisando y finalmente evaluando el uso de un modelo de Machine Learning basado y priorizado en un bajo costo

computacional para el reconocimiento de CAPTCHAS haciendo uso de una Red Neuronal Convolucional.

Dataset

Descripción

Para la implementación de este modelo, se utilizó un dataset obtenido mediante Kaggle, de nombre CAPTCHA Images [1], el cuál consiste de 1070 imágenes etiquetadas mediante el nombre de la imagen. Este dataset ya se encuentra con la mayor parte del preprocesamiento necesario para entrenar un modelo. Se encuentra, como se mencionó previamente, ya etiquetado, con todas las imágenes centradas, con el mismo formato y en blanco y negro, 5 letras por cada CAPTCHA, únicamente letras minúsculas y un tamaño fijo de 200 x 50.

Técnicas de escalamiento y aumentación

Debido a la poca cantidad de imágenes que proporciona este dataset, hice uso de la técnica de Data Augmentation con la herramienta de ImageDataGenerator de Keras, generando variaciones de las imágenes, para aumentar el tamaño del dataset. Se utilizó la siguiente configuración para el augmentation:

```python
train_datagen = ImageDataGenerator(
    rotation_range=6,
```

```
    brightness_range=[0.2, 1.6],
    width_shift_range=0.03,
    height_shift_range=0.03,
    zoom_range=0.2,
    shear_range=8,
)
```

Una vez obtenido el dataset, se realizó un EDA (Exploratory Data Analysis) para conocer un poco sobre las imágenes obtenidas, verificando el “formato” de ruido que tenían las imágenes, viendo que todas tenían básicamente lo mismo: Una línea con algunas deformaciones, un poco de distorsión y ruido visual en medio que distorsionaba en algunos casos algunas letras.

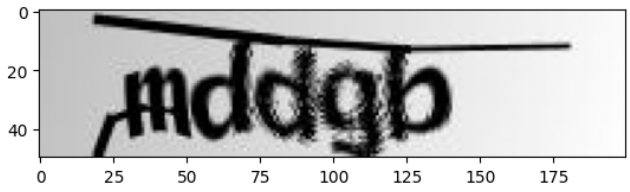

Figura 1. Ejemplo de un CAPTCHA

Viendo que básicamente todo estaba bien preprocesado, se procedió con la separación aleatoria de las imágenes, dividiendo primeramente un 80% del dataset para train, y el 20% para test, para posteriormente dividir también el train en 80% y 20% para train y validation respectivamente.

Análisis exploratorio

Con los datos ya debidamente separados, y continuando con nuestra exploración de las imágenes, se realizó una comparación entre el conteo de cada caracter posible de todos en todas las imágenes, con el fin de verificar un balance entre estos caracteres, para en caso de haber uno o más dominantes evitar un “overfitting” del modelo

sesgado hacia ese o esos caracteres. Y para hacerlo de una forma más fácil y visual se hizo una gráfica de barras como la que se muestra a continuación:

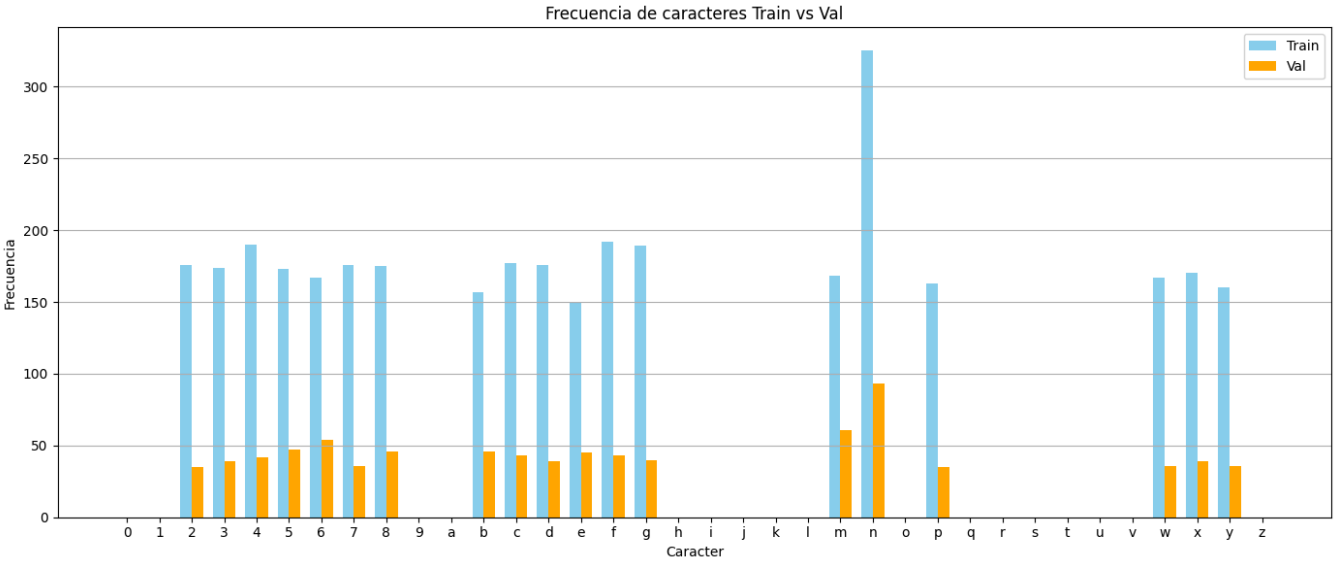

Figura 2. Gráfica de frecuencia de caracteres

De esta manera pude notar dos particularidades del dataset: Primero que no se encuentran presentes los números 0, 1 y 9, así como las letras a, h, i, j, k, l, q, r, s, t, u, v, y z, es decir una gran parte de los caracteres alfanuméricos posibles, dando únicamente 19 caracteres o clases para este caso posibles. Segundo que existe una letra dominante no solo para Train sino para Val también, así pues viendo que en el dataset completo en general existe esta tendencia de los CAPTCHAS a tener la letra n más que cualquier otra.

Balanceo de caracteres

Si bien para este caso en específico con este set de caracteres no perjudica el tener esta clase dominante ya que está presente en todo el dataset, sí podría ser mejor tratar de balancear las clases para evitar que el modelo “sobre prediga” la n. Para esto realicé una función que primero se encargue de realizar una lista de todas las imágenes que tengan al menos uno de los caracteres “raros” pero ninguna “n” o caracteres dominantes. Teniendo esta lista, la pasamos después a una función que se encarga de manera semi manual con ayuda del ImageDataGenerator a aumentar estas imágenes con caracteres raros, balanceando así nuestros caracteres (se repite

nuevamente si no se llega al threshold deseado). Cabe mencionar que al aumentar las imágenes y para tener una buena variación de estas, se tuvo que añadir un padding de 15 a cada imágen aumentada, y luego reajustar el tamaño de nuevo a nuestro 200x50, esto debido a que si queremos rotar en un buen rango, o mover en eje x e y, se ocupa el padding para que la imagen no se “salga” de sí misma, pues a veces al rotarla mucho podía tener caracteres fuera de la imagen.

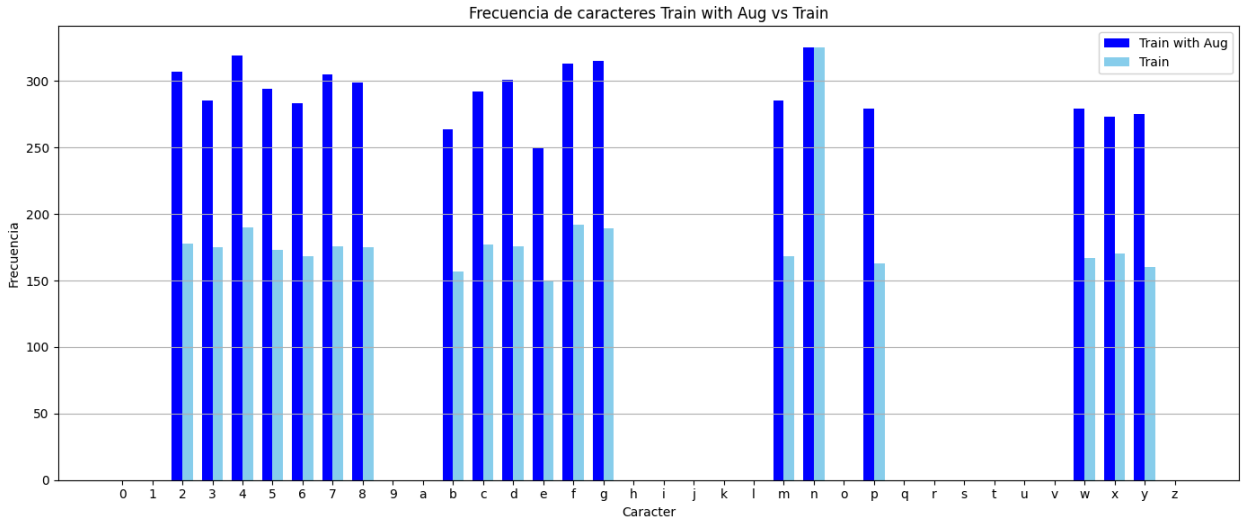

Figura 3. Gráfica de frecuencia de caracteres balanceados

Modelo CNN

Se decidió crear una Convolutional Neural Network (CNN) de acuerdo con el trabajo de Ning Yu y Kyle Darling [1]. Se escogió Tensor Flow ya que nos permite construir un modelo para esta red y nos da todas las facilidades necesarias, además por ser el framework aprendido en clase. También, permite el acceso directo a la creación de un modelo bien formado antes de compilar.

En primer lugar, se establece una entrada constante para la red: se decidió utilizar el tamaño de entrada fijo que ya venía de las imágenes de 200x50 píxeles. Utilizando esto como base principal de lo que construir para nuestra red, se construyó un modelo

de Red Neuronal Convolucional como se muestra en la Figura 4. Se escogió un batch de 64, una época “inicial” de 40 (porque hubo más entrenamiento, en suma fueron 70 épocas de entrenamiento), una tasa de dropout de 0.25 y una tasa de aprendizaje de 0.001. Se utilizaron como métricas las recomendadas para este modelo según Ning Yu y Kyle Darling: accuracy y cross entropy.

Figura 4. Tabla descriptiva de la CNN

Capa	Parámetros
Convolucional (Conv2D)	16 filtros 3x3 kernel kernel reg 0.02 ReLU activation
Conv2D	32 filtros 3x3 kernel ReLU activation
MaxPooling2D	2x2
Dropout	0.25
Flatten	NA
Dense	64 filtros kernel reg 0.02

	ReLU activation
Dense (OutChar1)	19 clases Softmax activation
Dense (OutChar2)	19 clases Softmax activation
Dense (OutChar3)	19 clases Softmax activation
Dense (OutChar4)	19 clases Softmax activation
Dense (OutChar5)	19 clases Softmax activation

Dada la naturaleza del modelo y el dataset, se tenían 5 capas densas al final, una por cada caracter a predecir con 19 caracteres o clases posibles, con activación softmax para la predicción, siendo por este mismo motivo que se utilizó la función de “sparse categorical crossentropy” como loss, ya que se hizo un label encoding para las etiquetas de nuestros CAPTCHAS.

Resultados

Primer modelo

Si bien se utilizó finalmente la arquitectura descrita anteriormente en la Fig 4, inicialmente no fue de esta manera, pues la propuesta por el paper utilizado usaba una capa densa más antes de las capas finales de predicción, y contemplaba un dropout de 0.2 a diferencia del 0.25 que se usó finalmente. Para esto entonces el primer modelo desarrollado consistió de la siguiente manera:

Figura 5. Tabla descriptiva del primer CNN

Capa	Parámetros
Convolucional (Conv2D)	32filtros 3x3 kernel ReLU activation
Conv2D	64filtros 3x3 kernel ReLU activation
MaxPooling2D	2x2
Dropout	0.20
Flatten	NA
Dense	128 filtros ReLU activation
Dense	64 filtros ReLU activation
Dense (OutChar1)	19 clases Softmax activation
Dense (OutChar2)	19 clases Softmax activation
Dense (OutChar3)	19 clases Softmax activation

Lamentablemente este modelo no dió los resultados deseados o siquiera buenos, pues además de tener un alto loss y bajo accuracy para validation, también lo tenía para train, aunque en menor medida:

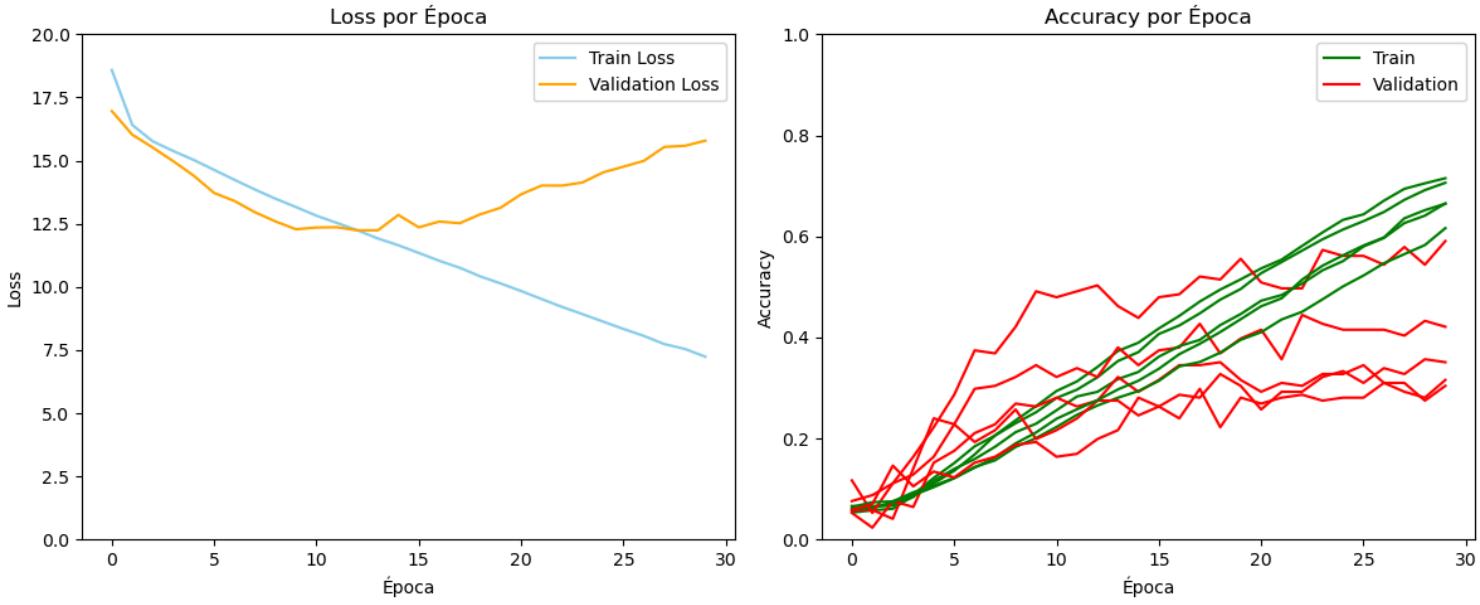

Figura 6. Gráficas del primer modelo

Con esto además de observar el mal resultado obtenido tanto para val como para train, pues train apenas llega a un accuracy de ~0.75 en el mejor de los casos (char 1) y un

loss de 7.5, también se puede observar un claro caso de overfiting dado que el loss para validation sin bien descendió un poco, aproximadamente para la época 10 se estancó y en la 13 empezó a subir de nuevo, pero la de train no. Esto indicaría que el modelo iba aprendiendo bien los patrones pero solamente para el set de train, equivocándose cada vez más para validation. Y lo mismo se ve para el accuracy, pues en el mejor de los casos para val, solo llegó al 0.6.

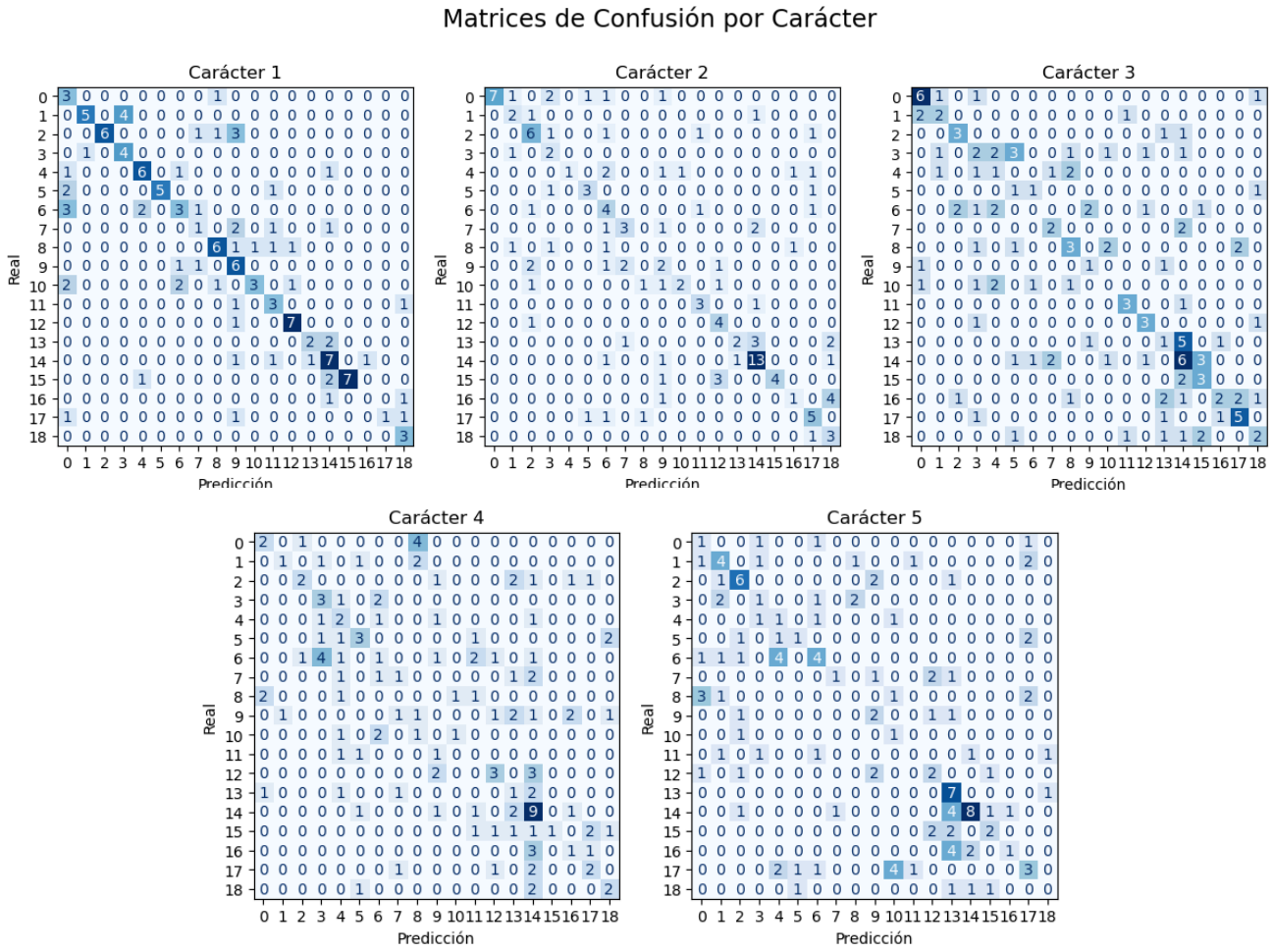

Figura 7. Matrices de confusión del primer modelo

Asimismo con las matrices de confusión podemos observar un muy mal desempeño para la mayoría de los caracteres, excepto un poco el caracter 1 que podemos ver obtiene las mejores predicciones.

Segundo modelo

Buscando reducir el mal desempeño en val, es decir evitar o reducir el overfitting, se continuó con las mismas características del modelo anterior, pero únicamente añadiendo “kernel regularizers” que sirven para penalizar valores grandes de los pesos (kernels) durante el entrenamiento, mejorando la capacidad de generalización del modelo. Se estableció el coeficiente de regularización a 0.01, que es básicamente qué tanto va a afectar el regularizador al loss durante el entrenamiento, es decir que a mayor coeficiente mayor penalización a los pesos. Estos regularizadores se añadieron a las 2 capas convolucionales así como a las 2 densas (recordando que venimos con la arquitectura descrita en la Fig. 5). Con el mismo objetivo, también se aumentó el dropout a 0.30 y se añadió otra capa de dropout entre las dos primeras capas convolucionales, lo que quiere decir que durante cada iteración de entrenamiento, el 30% de las salidas o activaciones se “neutralizan” (o se multiplican por 0) aleatoriamente de la capa que precede al dropout.

Y con estos cambios a nuestro modelo obtuvimos los siguientes resultados:

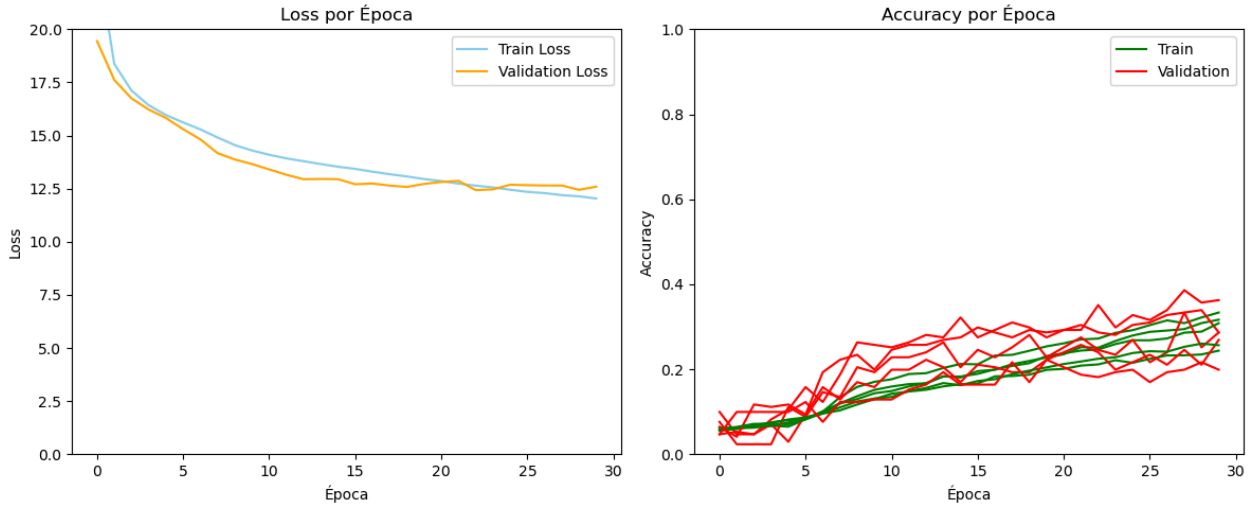

Figura 8. Gráficas del segundo modelo

Analizando esto podemos notar que en efecto conseguimos reducir el overfitting puesto que ahora tanto el loss como el accuracy van a la par para val y tran, sin embargo

ahora hay un underfit pues ahora tienen peores resultados que antes, dando a entender que no está aprendiendo bien el modelo.

Caracter	Mejor acc (train/época)	Mejor acc (val/época)	Última acc (train)	Última acc (val)	Promedio acc (train)	Promedio acc (val)
Char 1	0.3165, 30	0.3860, 28	0.3165	0.3626	0.1873	0.2495
Char 2	0.3333, 30	0.3392, 29	0.3333	0.2865	0.2021	0.2347
Char 3	0.3082, 30	0.3333, 28	0.3082	0.2865	0.182	0.2008
Char 4	0.2437, 30	0.2690, 30	0.2437	0.269	0.1596	0.1832
Char 5	0.2602, 29	0.2222, 20	0.2565	0.1988	0.1654	0.1565

Menor loss (entrenamiento): 12.0345 en la época 30
Menor val_loss (validación): 12.4284 en la época 23

Figura 9. Resultados del segundo modelo

Como podemos ver, se obtuvieron pésimos resultados de este modelo, teniendo como mejor accuracy para train un 0.31 en el char 1, y un 0.38 para validation igual en char 1, denotando que incluso va “mejor” para el set de validation que el train mismo, o sea que aprendió muy poco, pero bien a generalizar, que es justo lo que buscábamos.

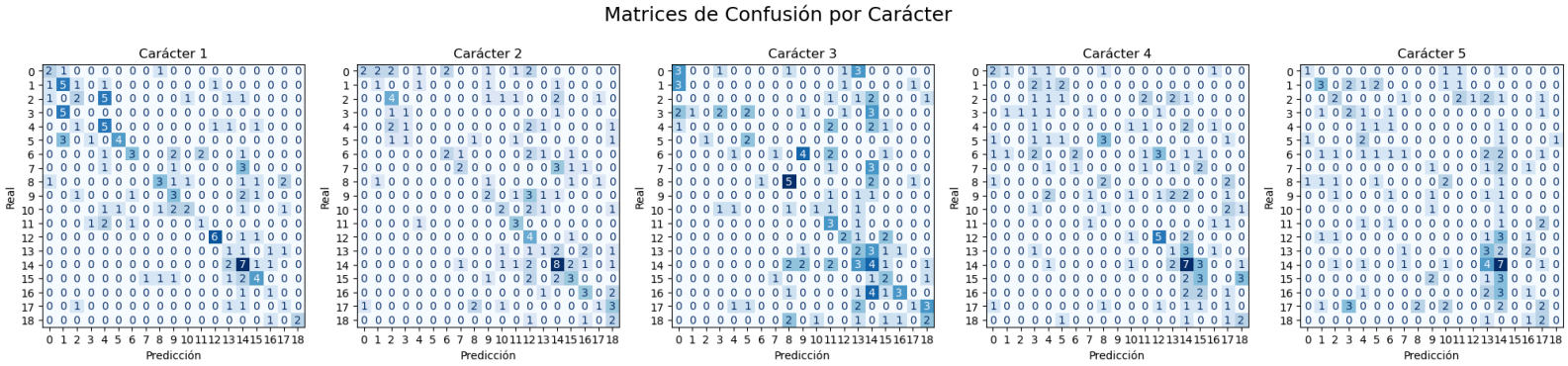

Figura 10. Matrices de confusión del segundo modelo

Y la historia se repite para este modelo, observando un muy mal desempeño en este caso para todos los caracteres, aunque igualmente el caracter 1 podemos ver obtiene mejores predicciones, que viendo que se repite podríamos inferir que se debe a que al ser el primero, es más fácil reconocerlo como objeto para después predecir su valor, siendo los demás un poco más complicados ya que se pueden “perder” entre todos.

Tercer modelo

Ahora que tenemos un poco mejor el modelo en cuanto a overfit se refiere, intentamos ajustar hiperparametros del modelo buscando ahora mejorar el aprendizaje, y después de algunos intentos y ajustes, se llegó al modelo descrito por la Figura 4, quitando una capa densa, ajustando los coeficientes y el dropout, consiguiendo entonces los mejores resultados que pude obtener:

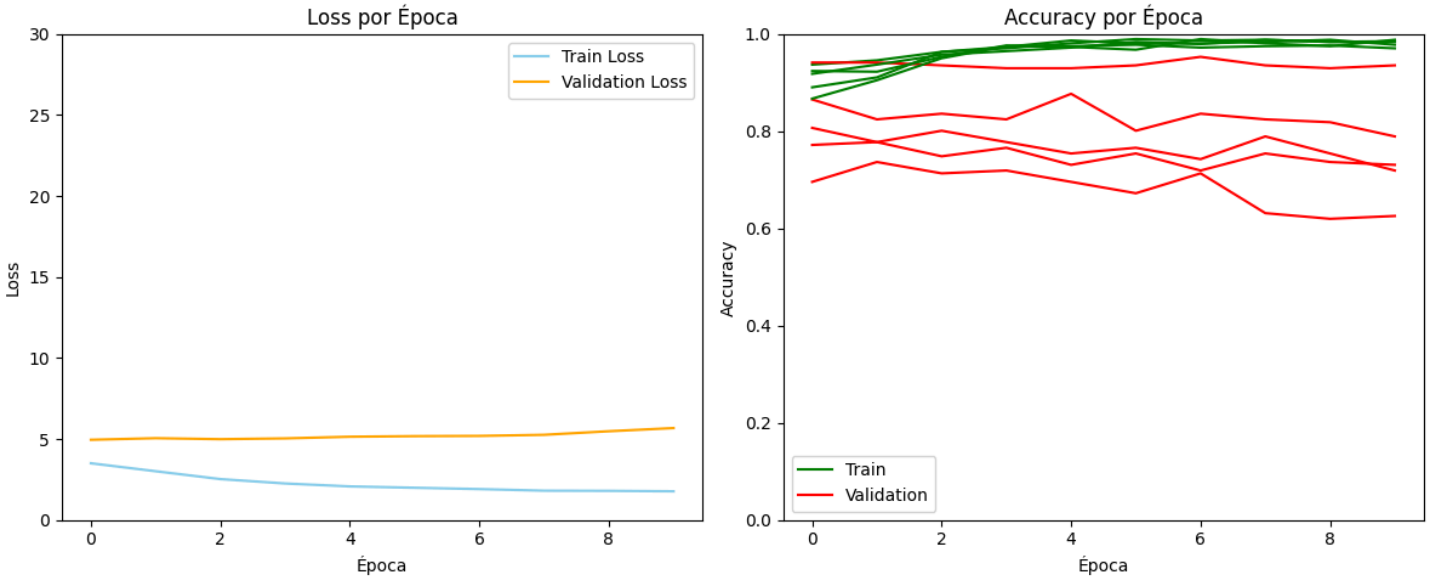

Figura 11. Gráficas del tercer modelo

(últimas 10 épocas)

Caracter	Mejor acc (train/época)	Mejor acc (val/época)	Última acc (train)	Última acc (val)	Promedio acc (train)	Promedio acc (val)
Char 1	0.9898, 6	0.9532, 7	0.9839	0.9357	0.9731	0.9368
Char 2	0.9883, 8	0.8772, 5	0.9854	0.7895	0.9672	0.8298
Char 3	0.9883, 9	0.7778, 2	0.9781	0.731	0.9638	0.7491
Char 4	0.9781, 6	0.7368, 2	0.9708	0.6257	0.9546	0.6825
Char 5	0.9898, 7	0.8070, 1	0.9883	0.7193	0.9653	0.769

```
Menor loss (entrenamiento): 1.7737 en la época 10
Menor val_loss (validación): 4.9558 en la época 1
```

Figura 12. Resultados del tercer modelo

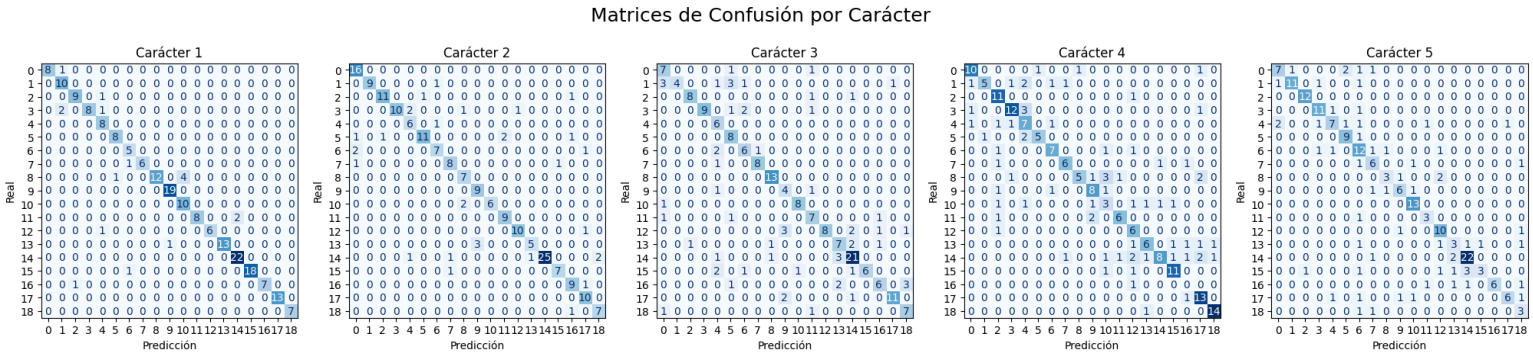

Figura 13. Matrices de confusión del tercer modelo

Observamos que si bien no tenemos unos accuracys tan bueno como el del paper principal utilizado, esto se debe a que, según infiero, el paper utiliza un modelo de OCR y segmentación para primero detectar la ubicación de los caracteres y después poder predecir con ello cada caracter. Sin embargo me parece que comparando con el estado del arte, dado que es un modelo muy simple de CNN que realmente es muy barato computacionalmente hablando (~5,000,000 de parámetros con 1,070 instancias), obtiene un muy buen rendimiento en general para test.

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s
Test Loss: 4.9023
Accuracy char1: 0.9206
Accuracy char2: 0.8692
Accuracy char3: 0.7664
Accuracy char4: 0.7290
Accuracy char5: 0.7570
```

Figura 14. Evolución del tercer modelo

Referencias

- Yu, N., & Darling, K. (2019). A low-cost approach to crack Python CAPTCHAs using AI-based chosen-plaintext attack. Applied Sciences, 9(10), 2010. https://doi.org/10.3390/app9102010
- Wang, J., Qin, J. H., Xiang, X. Y., Tan, Y., & Pan, N. (2019). CAPTCHA recognition based on deep convolutional neural network. Mathematical Biosciences and Engineering, 16(5), 5851–5861. https://doi.org/10.3934/mbe.2019292