Aprendizaje Automático - Tarea n°2

Vicente Mieres

***Resumen*—Este informe presenta el desarrollo, entrenamiento y evaluación de modelos de MLP, es decir *Multi-layer Perceptron*, para la clasificación de imágenes de dos subconjuntos pertencientes a QuickDraw: QuickDraw-10 y QuickDraw-Animals. Se entrenaron tres configuraciones distintas de MLP, variando el número de capas ocultas, funciones de activación y funciones de pérdida, siguiendo los criterios establecidos. Cada modelo fue evaluado mediante cinco repeticiones y con pesos inicializados aleatoriamente, para así, asegurar la robustez de los resultados. Se reportaron métricas de precisión (accuracy) por clase y global, junto con matrices de confusión, asociadas a cada modelo. Finalmente, se analiza la toma de decisiones en lo que respecta a diseño, y como esto influye en el desempeño de los modelos, esto para poder ofrecer recomendaciones a trabajos futuros.**

***Index Terms*—MLP, loss function, activation function, pesos, capas ocultas.**

I. Introducción

La clasificación automática de imágenes es un problema central en el campo del aprendizaje automático y la inteligencia artificial, areas que han experimentado un claro auge en las últimas décadas. En este contexto, los modelos MLP han demostrado ser una herramienta efectiva para abordar tareas de clasificación supervisada, estos debido a la capacidad de modelar relaciones no lineales en los datos. Así, el conjunto de datos QuickDraw, el cual contiene millones de dibujos a mano alzada en diversas categorías, proporciona una amplia cantidad de imágenes de prueba para evaluar la capacidad de aprendizaje y generalización de estos modelos.

Este trabajo se enfoca en entrenar y evaluar tres configuraciones diferentes de MLP utilizando dos subconjuntos representativos del conjunto QuickDraw: QuickDraw-10, que contiene 10 clases seleccionadas, y QuickDraw-Animals, que agrupa diversas categorías relacionadas con animales. El objetivo principal es analizar cómo las decisiones de diseño, como el número de capas ocultas, las funciones de activación y las funciones de pérdida, afectan el desempeño de los modelos en términos de *accuracy*, tanto por clase como general del modelo. De esta manera, se busca obtener una comprensión más profunda de las características que influyen en la efectividad de las redes neuronales para la clasificación de imágenes simples y rápidas.

II. Desarrollo

En la siguiente sección se detalla el procedimiento utilizado para la correcta implementación del programa. La Figura 1 muestra un diagrama de bloques con la solución a grandes rasgos.

II-A. Preprocesamiento de Datos

Como paso inicial en la construcción de los modelos, se llevó a cabo la carga y procesamiento preliminar de los datos para el subconjunto QuickDraw-10. Este proceso comenzó con la lectura de los archivos train.txt y test.txt, con el fin de obtener cuatro listas diferenciadas: dos de ellas contienen las rutas de las imágenes, mientras que las otras dos almacenan las etiquetas correspondientes a cada imagen. Para esto se utilizó la función **load_image_paths_and_labels()**, la cual, al recibir la ruta a un archivo, retorna dos arreglos separados: uno con las rutas de las imágenes y otro con sus etiquetas asociadas.

Posteriormente, mediante la función **load_images()**, que recibe como parámetro el listado de rutas, se procedió a la lectura y carga de las imágenes. Cada imagen fue transformada mediante un aplanamiento a un vector unidimensional de longitud 65.536 (mediante reshape(-1)), con el fin de adecuarlas al formato requerido para la entrada de los modelos MLP.

De forma similar, para el subconjunto QuickDraw-Animals, se utilizó la función **load_image_images_and_labels()**, donde la única diferencia significativa es que se leen las imágenes directamente desde su carpeta de originen, y no se utiliza el *path* a la imagen, este cambio se debe a la diferencia de formato en que se presentan los subconjuntos. Tras lo mencionado, se obtienen tanto los conjuntos de train, test y las etiquetas correspondientes. Cabe destacar que las funciones mencionadas en esta sección fueron construidas utilizando I.A. generativa para agilizar el proceso de codificación [1].

Una vez leídas las imágenes, se realizaron tres modificaciones. Se separó el conjunto de entrenamiento en dos, esto para obtener un conjunto de validación, extrayendo el 15 % de los datos originales de *train*, luego, se normalizaron los datos, simplemente realizando una división por 255. Y finalmente, se hizo *one hot enconding* para las etiquetas, es decir, convertidas a etiquetas categóricas, esto para que el algoritmo de MLP sea mas eficiente al trabajar con datos binarios.

II-B. Clase MLP y funciones principales

Una vez preparados los datos de entrada, se deben construir los modelos, para esto se utilizará la clase *Model*, perteneciente a la librería *tensorflow*, para implementar la clase propia *MLP()*; destacando atributos propios de un perceptrón multicapa, tales como la función de activación, número de capas, entre otros. Además de esta implementación, se han construido tres funciones. La primera de estas, **train()**, es la encargada de entrenar un modelo con diferente parámetros, tales como la función de pérdida, el número de épocas o el tamaño de los *batchs* (lotes), además se consideró para esta función *EarlyStopping*, es decir que detiene el entrenamiento si luego de 5 épocas no se obtiene una mejora, particularmente luego de evaluar el conjunto de validación. Luego, la función **evaluate()**, tal como su nombre indica, evalúa el modelo, obteniendo las predicciones realizadas por el mismo y entregando como resultado tanto el *accuracy* total del modelo como el de cada

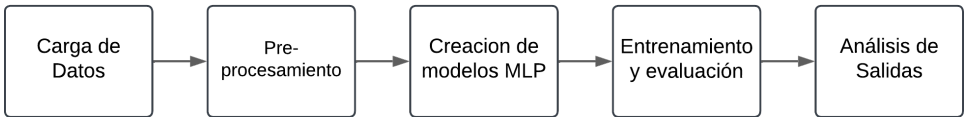


Figura 1: Diagrama de bloques

clase, y además la matriz de confusión respectiva. Por último, **experiment()** se encarga de realizar el objetivo principal de esta tarea, esta toma todos los datos necesarios, tales como datos de entrenamiento, test, validación, número de épocas, entre otros; y utiliza las dos funciones mencionadas antes para realizar *n* experimentos, es decir n entrenamientos de un modelo en particular y entregar listados con *accuracy* total, por clase y las correspondientes matrices de confusión. Para un entendimiento total del funcionamiento del programa, revisar el código fuente.

II-C. Construcción de modelos

Utilizando las funciones antes mencionadas, en especifico con **experiment()**, se construyeron tres modelos con diferentes características, así la idea principal de la solución al problema planteado es crear modelos que escalen en tanto en capas, lotes y épocas, y además se considerarán diferentes combinaciones de funciones de activación y pérdida. Finalmente se obtendrá un total de tres modelos para cada conjunto de datos. El listado de modelos y sus características se encuentra representado en la Tabla I. En particular, los valores escogidos para la construcción de los modelos se determinaron en función de lo mencionado por GeeksforGeeks [2], es decir, un modelo pequeño, mediano y grande, esto en lo que refiere a épocas y lotes. Además, para tener mayor variabilidad entre los modelos, se optó por tener dos, tres y cuatro capas, en los respectivos modelos uno, dos y tres.

Finalmente, como función de activación se escogieron *sigmoid* y *tanh*, esto pues, sigmoid corresponde a una de las más utilizadas, y dada una correcta transformación de los datos es posible utilizarla en contextos multiclase, mientras que tanh, si bien trbaja similar a sigmoid, esta presenta un rango más amplio en los datos, entre -1 y 1, y no 0 a 1, por lo que tiende a reducior el sesgo. Por otro lado, como función de pérdida se optó por *categorical_crossentropy* y *categorical_hinge*, si bien la primera de estas es adecuada para modelos multiclase, principalmente cuando más de dos clases son mutuamente excluyentes. El caso de hinge es diferente, esta es utilizada en modelo SVM principalmente, es por esta razón fue escogida, para determinar el comportamiento dado un modelo diferente para el que fue diseñada.

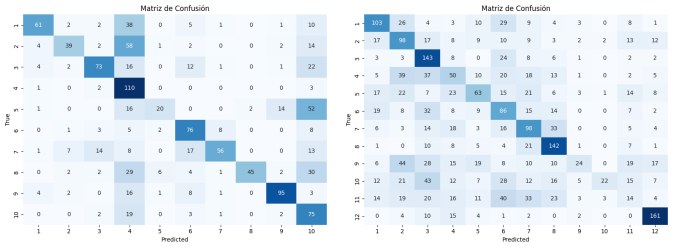
III. Análisis de Resultados

En primer lugar, cabe destacar que para el proceso en su totalidad se utilizaron ambos *dataset* completos, es decir, todas las imágenes de entrenamiento y evaluación entregadas para esta tarea. Esto bajo un procesador AMD Ryzen7 de la serie 5000, y 16GB de memoria RAM, utilizando CPU.

De esta forma, se presentan los resultado obtenidos para cada uno de los modelos. Puesto que se han realizado cinco entrenamientos por cada uno, la Tabla II muestra la precisión del modelo correspondiente a la mediana. Donde el modelo con mejor rendimiento corresponde al primero, utilizando el subconjunto QuickDraw-10, sin embargo, este posee un *acc* bastante malo, cercano a lo que sería tirar una moneda, aproximadamente 50 %, por lo que no es un valor bueno. Por otro lado, el resto de modelo presentan aun peores resultados, bordeando el 20 % de precisión en la mayoría de estos, menos en el modelo **A1**, correspondiente al primero de **QD1**, el cual bordea el 40 % de precisión. Las posibles razones de estos resultados son discutidas en las subsecciones siguientes.

III-A. Modelo 1

A continuación se realizará un análisis para el modelo n°1, es decir **QD1** y **A1**, mostrando una comparación según cada conjunto de datos. En primer lugar es posible asegurar que, aunque la precisión de ambos no es esperable, el modelo **QD1** posee mejor capacidad para generalizar que el modelo **A1**, lo cual se puede deber principalmente al número de clases, puesto existe una diferencia de 2 entre ambos conjuntos de clases. Por otro lado, las Figuras 2 y 3 muestran las predicciones por clase y las matrices de confusión de cada modelo, en estas es posible apreciar que el modelo **QD1** tiende a predecir de forma mas acertada la clase 4, y menor medida pero no menor las clases 9 y 10, mientras que para el caso de **A1** este predice casi de forma perfecta la clase 12, y obteniendo así un *acc* muy bajo para las clases 9, 10 y 11, notando diferencias en el comportamiento de un modelo para conjuntos de datos totalmente distintos, por lo cual son totalmente validos estos resultados, aunque no ideales. Finalmente, en cuanto a los hyperparámetros del modelo, es posible que se tenga problema con la función de activación, puesto que está diseñada para trabajar con valores binarios, y por mas que se esté realizando *one hot coding* a las etiquetas, esta puede estar trabajando de manera sub-optima. Por otro lado, se tomarán como validos las cantidades de épocas y lotes para este modelo, y serán analizadas en las siguientes secciones.



(a) Matriz de Confusión QD1 (b) Matriz de Confusión A1

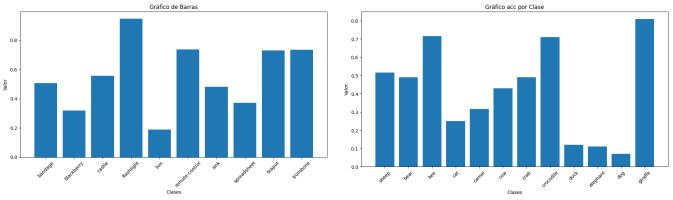
Figura 2: Matriz de Confusión Modelo 1

Tabla I: Modelos MLP

Modelo	Capas	Tamaño lote	Épocas	Fn Activación	Fn Pérdida
QD1	[512,256]	32	20	sigmoid	categorical_crossentropy
QD2	[512, 256, 128]	128	50	sigmoid	categorica_hinge
QD3	[512, 256, 128, 64]	512	100	tanh	categorica_hinge
A1	[512,256]	32	20	sigmoid	categorical_crossentropy
A2	[512, 256, 128]	128	50	sigmoid	categorica_hinge
A3	[512, 256, 128, 64]	512	100	tanh	categorica_hinge

Tabla II: Resultados precisión por modelo

Modelo	Precisión
QD1	0.5574614065180102
QD2	0.2032590051457976
QD3	0.12521440823327615
A1	0.4185077115464777
A2	0.21717382242601083
A3	0.09462275948311796



(a) *Accuracy* QD1 (b) *Accuracy* A1

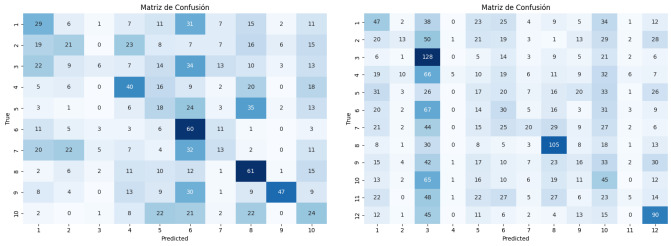
Figura 3: Precisión por Clase Modelo 1

III-B. Modelo 2

Tal como se muestra en la tabla II, los resultados obtenidos por lo modelos **QD2** y **A2** son muy desalentadores, ambos alrededor del 20 % de precisión. Observando las Figuras 4 y 5, es posible asegurar que la capacidad de generalización de modelo es derechamente mala, en especial el modelo **A2**. Este rendimiento bajo, sugiere claramente que el modelo no está aprendiendo las características de los datos, además las matrices de confusión presentan alta cantidad de falsos negativos y falsos positivos, demostrando dificultad para distinguir entre clases.

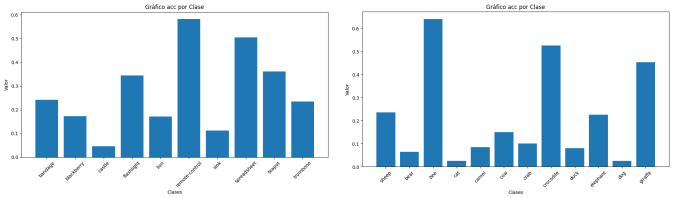
Por otro lado, un punto no mencionado para el modelo 1 (dado el bajo impacto), es el hecho de utilizar EarlyStopping, es decir, el entrenamiento del modelo se detendrá tras n número de épocas, bajo el criterio de no encontrar mejor en alguna característica. En particular para los tres modelos se establecieron cinco épocas y este se detendrá en función de la pérdida obtenida en el conjunto de validación.

Tras lo mencionado, y la similitud en rendimiento entre ambos modelos, es posible concluir que las configuraciones escogidas para este modelo no son las adecuadas, principalmente la función de pérdida *categorical_hinge*, la cual, tal como se menciona anteriormente, es utilizada en la mayoría de los casos en modelos SVM y no MLP, por lo que un cambio en esta podría mejorar la presición en general. Lo mismo se podría decir para la función sigmoid, utilizar alguna otra como **ReLU** o **Leaky ReLU**, podría ayudar a mejorar el rendimiento general del modelo. Por ultimo, un aumento en la cantidad de épocas sin mejora, por ejemplo a diez, proporcionaría más libertad al modelo para aprender de mejor forma sobre los datos.



(a) Matriz de Confusión QD2 (b) Matriz de Confusión A2

Figura 4: Matriz de Confusión Modelo 2



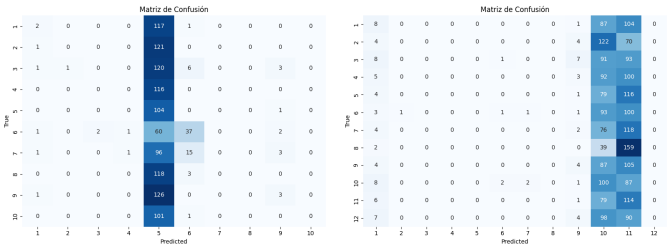
(a) *Accuracy* QD2 (b) *Accuracy* A2

Figura 5: Precisión por Clase Modelo 2

III-C. Modelo 3

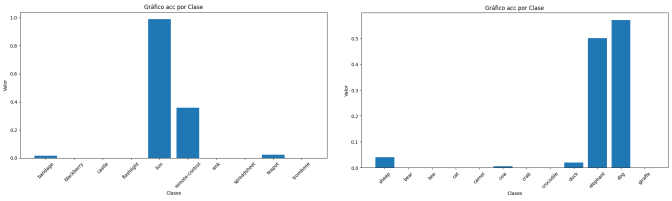
Los resultados obtenidos para el modelo 3, es decir **QD3** y **A3**, poseen un rendimiento totalmente descartable, a tal punto que el segundo de estos no supera el 10 % de *acc*, múltiples razones pueden producir este fenómeno, y estas ya fueron mencionadas para los casos anteriores, sin embargo no se encontraban todas centralizadas en un solo modelo. Por lo que, tal como en el modelo 2, la capacidad de predecir las clases de este no es para nada esperable, de tal forma que al observar las Figuras 6 y 7, es posible notar que solo se está prediciendo uno o dos clases respectivamente, es decir que no está aprendiendo a distinguir entre categorías. Además, tal como en el modelo anterior, la estructura escogida para este modelo no es ideal, nuevamente el uso de *categorical_hinge* como función de pérdida está demostrando una clara falencia a la hora de construir el modelo, y el hecho de tener tan pocas épocas en lo que sería EarlyStopping, está produciendo que el modelo no termine de realizar un entrenamiento satisfactorio. Por ultimo, un punto no mencionado antes, es que a medida que el tamaño de las capas aumenta, en conjunto con el numero de épocas y tamaño de los lotes, el rendimiento disminuye. Es por todo esto que se recomienda realizar un búsqueda más exhaustiva de todos los hiperparámetros del

modelo, puesto que los resultados obtenidos no son para nada buenos.



(a) Matriz de Confusión QD3 (b) Matriz de Confusión A3

Figura 6: Matriz de Confusión Modelo 3



(a) *Accuracy* QD3 (b) *Accuracy* A3

Figura 7: Precisión por Clase Modelo 3

IV. Conclusión

Este informe presentó el entrenamiento y evaluación de tres configuraciones de modelos MLP para la clasificación de imágenes utilizando los subconjuntos QuickDraw-10 y QuickDraw-Animals. Los modelos, aunque entrenados con diferentes combinaciones de funciones de activación y pérdida, mostraron resultados decepcionantes. El primer modelo, **QD1** y **A1**, tuvieron un rendimiento relativamente mejor, alcanzando una precisión del 55.7 % en QuickDraw-10, pero aún con una alta variabilidad y dificultades para generalizar, mientras que los modelos posteriores **QD2**, **QD3**, **A2** y **A3** presentaron precisiones alrededor del 20 %, o incluso menores.

Los resultados sugieren que el uso de Sigmoid y Categorical Hinge como funciones de activación y pérdida respectivamente no fueron las más adecuadas para clasificación multiclase. Se recomienda reemplazar Sigmoid en la capa de salida y explorar funciones de activación como ReLU o Leaky ReLU en las capas ocultas. Además, la aplicación de técnicas de búsqueda de hiperparámetros deberían mejorar la capacidad de generalización del modelo.

En conclusión, aunque los modelos de MLP pueden ser efectivos en tareas de clasificación, los resultados obtenidos en este estudio destacan la importancia de seleccionar adecuadamente los hiperparámetros, funciones de activación y funciones de pérdida para optimizar el rendimiento en problemas de clasificación multiclase.

Referencias

[1] ChatGPT, “Asistencia generativa para codificación de modelos mlp,” 2025. [Online]. Available: https://chat.openai.com/

[2] GeeksforGeeks, “How to choose batch size and number of epochs when fitting a model?” 2025. [Online]. Available: https://www.geeksforgeeks.org/how-to-choose-batch-size-and-number-of-epochs-when-fitting-a-model/