Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Comparación de modelos de Machine Learning para la clasificación de imágenes. ViT vs. gMLP

Machine Learning models comparision for image classification. ViT vs. gMLP

Hernán Daniel González Guanipa

La Laguna, 13 de septiembre de 2022

D. **Sergio Díaz González**, con N.I.F. 7906162-D profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43826207-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistema de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Comparación de modelos de Machine Learning para la clasificación de imágenes. ViT vs. gMLP*”.

ha sido realizada bajo su dirección por D. **Hernán Daniel González Guanipa**, con N.I.F. 43851135-W.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de septiembre de 2022.

Agradecimientos

Debo agradecer a los profesores Sergio Díaz González y Jesús Miguel Torres Jorge por hacer posible la finalización de este proyecto y por darme una oportunidad de demostrar lo que puedo hacer.

A mis padres y mi hermana por haber hecho enormes esfuerzos y sacrificios para llegar hasta donde están y con ello alentarme a mi a seguir siempre adelante pase lo que pase.

A mi pareja, por haberme ayudado y apoyado incondicionalmente, siempre con un cariño indescriptible.

A mis amigos por ser uno de los pilares por los que hoy soy como soy.

Y por último, a las amistades que hice a lo largo de la carrera por alentarme a no rendirme y con los que ahora tengo lazos para toda la vida.

Licencia

Resumen

La producción de hardware más avanzado y potente está correlacionada con el desarrollo y puesta en práctica de conceptos teorizados anteriormente sobre la Inteligencia Artificial. Uno de estos conceptos que toma cada vez más importancia en esta era es el Deep Learning, que busca dotar de la capacidad de aprendizaje a las máquinas con o sin interacción humana de por medio. Con esta capacidad se puede traducir texto, predecir valores según los ya conocidos o clasificar imágenes, entre muchas otras aplicaciones. En este trabajo se comparan dos modelos de Deep Learning que tienen el objetivo de catalogar imágenes según las características que aprendan los modelos sobre estas. Los dos modelos se lanzaron el mismo año y son similares respecto a las bases, pero diferentes en su mecanismo principal. Por eso se busca obtener resultados concluyentes donde pueda aclararse qué modelo es más efectivo para cada conjunto de datos, yendo estos desde imágenes sencillas como números del 0 al 9 escritos a mano, hasta conjuntos más complejos como los sentimientos de una persona por su expresión facial.

Palabras clave: Deep Learning, Multilayer Perceptron, Vision Transformer, gMLP, Atención, Dataset, Neurona

Abstract

The most advanced and powerful hardware production is correlated with the development and implementation of previously theorized concepts about Artificial Intelligence. One of these concepts that take more and more importance in this era is Deep Learning, which seeks to provide machines the ability to learn with or without human interaction involved. With this ability you can translate text, predict values according to the already known or classify images among many other applications. In this end-of-degree project, we are going to compare two Deep Learning computer vision models that have the objective of classifying images according to the attributes that these models learn about every image. Both models were published in the same year, and they have similar basis, but their main mechanisms are different. This is why we are looking for conclusive results where it can be clarified which model is more effective depending on the dataset. Starting from easy images like numbers from zero to nine written by hand, to more complex datasets such as a person's feelings for their facial expression.

Keywords: Deep Learning, Multilayer Perceptron, Vision Transformer, gMLP, Attention, Dataset, Neuron

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Desde hace unos años el concepto de Inteligencia Artificial ha vuelto a tomar partido en las investigaciones y el desarrollo en todo el mundo, después de haberse dejado de lado por un largo periodo, el cual es llamado el ‘Invierno de las IA’.

No obstante, la irrupción de un gran salto tecnológico, ha permitido abrir muchas puertas en este campo, dotando a las máquinas de cierto grado de "inteligencia". Esto se consigue dándoles la capacidad de aprender partiendo de algoritmos, los cuales pueden ser modificados en el proceso de aprendizaje con unos datos de entrenamiento. Con ello se persigue el objetivo de crear predicciones, clasificar o identificar patrones sobre grandes conjuntos de valores, teniendo en mente que la máquina genere resultados cada vez más precisos. A esto se le denomina como Machine Learning o aprendizaje automático.

Dentro del Machine Learning se encuentran tres tipos de aprendizaje: Supervisado, no supervisado y por refuerzo. En el primero, y al que más importancia se le dará en este documento, el computador aprende con los datos introducidos previamente etiquetados por humanos, de manera que la máquina reconfigure sus parámetros a lo largo de la ejecución para que la etiqueta previamente declarada por el usuario y la que genere el modelo sean la misma. De esta manera el algoritmo entrena con un “histórico” de datos y aprende a predecir las etiquetas o valores de salida para un conjunto de datos diferente, como ejemplo se puede ver un algoritmo para clasificar imágenes en perros o gatos, recomendar películas según las ya vistas, marcar un correo como “Spam” o no, etc. El segundo tipo de aprendizaje es también pasándole unos datos de entrada, pero sin etiquetar, de manera que el propio algoritmo va a agrupar los valores y generará un conjunto de características que aprendió por sí solo para saber juntar los datos con características similares. Por último, el tercer tipo de aprendizaje tiene como objetivo que el algoritmo aprenda de la propia experiencia, es decir, que ante una situación pueda elegir la mejor decisión después de un proceso de prueba y error, ejemplos de estos son el reconocimiento facial o clasificar secuencias ADN.

1.1. Objetivos

Con este trabajo se pretende explicar y comparar dos modelos basados en redes neuronales para la clasificación de imágenes, con el objetivo de dar una visión más clara de las diferencias entre ambos modelos y exponer el mejor para cada situación. Para ello primeramente se describe en qué mecanismos se basa cada modelo, dejando clara la diferencia entre ambos y a continuación se seleccionarán o generarán los conjuntos de imágenes que van a ser utilizados para entrenar los dos modelos; Estos irán desde conjuntos típicos como pueda ser imágenes de números escritos al alza hasta algunos más especializados y menos comunes como por ejemplo la catalogación de sentimientos con rostros de personas, con la intención de conseguir datos que permitan saber en qué campos o tipos de imagen es mejor cada modelo.

El siguiente paso será implementar y mejorar los modelos en Python con la librería para redes neuronales Keras, contenida en la biblioteca Tensorflow, para finalmente con los resultados de ejecutar estas implementaciones con los datasets elegidos, sacar conclusiones y comparativas que puedan esclarecer las diferencias entre ambos.

1.2. Plan de ejecución

Para conseguir los resultados que se buscan con este proyecto se establecen una serie de pasos a seguir, comenzando por lo básico, luego la aplicación y por último las conclusiones y mejoras, quedando los siguientes puntos:

I. Estudiar el estado del arte de técnicas de Deep Learning, concretamente Vision Transformer (ViT) y Gated Multilayer Perceptron (gMLP).

II. Obtener y procesar los datasets que se quieren utilizar para ambos modelos, de los cuales se obtendrán resultados de efectividad.

III. Estudiar las tecnologías e implementar los modelos para posteriormente entrenarlas.

IV. Evaluar los resultados, comparar y proponer mejoras.

Capítulo 2

Antecedentes y Estado del Arte

Existe una subrama del Machine Learning donde el objetivo es replicar la estructura de nuestro sistema nervioso, es decir, buscan asemejar redes de neuronas estructuradas por capas donde cada una de estas cumpla con tareas específicas y al unirlas todas den resultados para situaciones más complejas de lo usual e incluso con conceptos abstractos. A este subcampo se le denomina como Deep Learning.

2.1. Computer Vision

El intento de emular los sentidos humanos no es algo nuevo, creándose en 1960 el concepto de Computer Vision en las universidades pioneras en inteligencia artificial, y se empieza a experimentar maneras de captar las imágenes y que los computadores describieran lo que veían junto a las redes neuronales básicas que existían.

Pero no es hasta 1970 que se generan unos cimientos de los cuales partirían muchos de los algoritmos que hay hoy en día que permitirían extraer los bordes, conexiones entre objetos, etc. Y en la década de los 80 se presentó la primera versión de lo que sería el modelo más importante hasta el día de hoy para la clasificación de imágenes, las Redes Neuronales Convolucionales o CNN de Yann LeCun integrando el algoritmo de aprendizaje backpropagation [1] y se fueron originando versiones de este cada vez más complejas como Lenet-5.

Para dar apoyo a la creación de modelos cada vez mejores, apareció ImageNet en 2010 [2], una base de datos que se actualiza cada año con nuevas imágenes, hecha para ser usada en computer vision a gran escala. De la mano con esta base de datos se creó el desafío ILSVRC, buscando saber qué modelo de clasificación de imágenes es mejor.

En 2012 el ganador fue AlexNet que es una CNN más profunda y con más filtros que LeNet, teniendo un error en las 5 mejores predicciones del 15 %, el año siguiente ZFNet que también es una CNN con un error menor y así continuamente.

Por lo general, los modelos ganadores estaban apoyados en CNN o en otros tipos de redes como las RNN (Redes neuronales recurrentes), pero prevaleciendo en líneas generales el éxito de las redes neuronales convolucionales. Hasta que finalmente, en 2021 aparece un modelo que con mecanismos de atención que hace frente a las CNN, Vision Transformer y que se verá más en detalle en este trabajo.

2.2. Deep Learning

El componente más básico de los modelos de Deep Learning son las neuronas o perceptrones, estas reciben unos datos de entrada $(a_0, a_1, \ldots, a_n)$ unos pesos o importancias $(w_0, w_1, \ldots, w_n)$ para cada entrada de la propia neurona y un sesgo o bias (b) que indica que tan altos deben ser los pesos para que la salida de la neurona sea útil (este proceso es una función de regresión lineal) y se obtiene una salida.

$$RegresionLineal = a_0w_0 + a_1w_1 + \ldots + a_nw_n + b$$

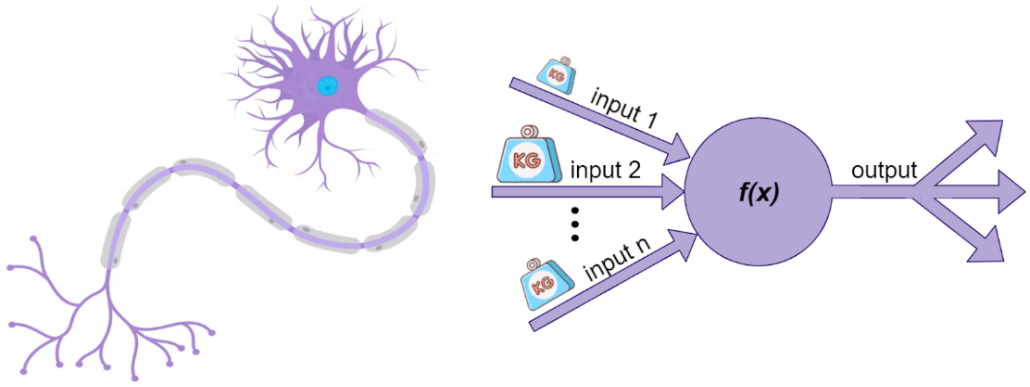

Figura 2.1: A la izquierda una neurona y a la derecha su interpretación en Deep Learning (Perceptron)

En estos algoritmos se crean sistemas donde la salida de las neuronas van unidas a otras neuronas, dando lugar a una red neuronal estructurada por capas. La primera de estas capas es la capa de entrada y corresponde al conjunto de neuronas que reciben los datos por primera vez. A continuación se encuentran las capas ocultas que puede ser una o más capas, sin existir un máximo de capas ocultas que pueda haber en la red, y por último la capa de salida que son las neuronas que dan el resultado de la red.

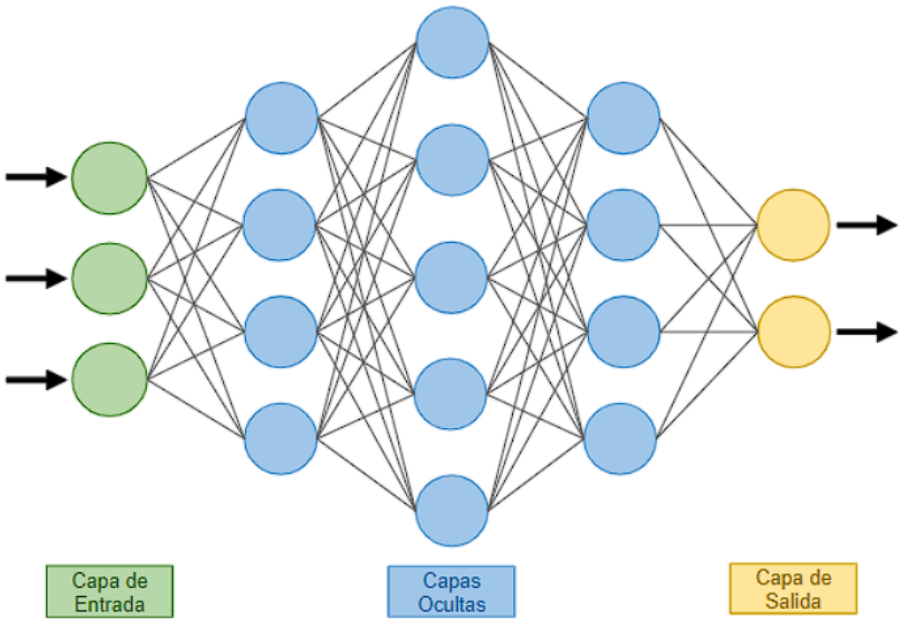

Figura 2.2: Red neuronal

Al crear una red neuronal de este tipo y sin ninguna otra herramienta, se da un cuello de botella que resulta en haber ejecutado una red con 3 neuronas únicamente, es decir, aunque una red esté hecha de 100 neuronas. Por ejemplo, así como está, el resultado será el de aplicar la regresión línea con cada neurona a los datos de entrada, será el mismo que haber pasado dichos datos por tres neuronas, la de entrada, una neurona que corresponde a la capa oculta y la de salida. Esto se debe a que al sumar los resultados de muchas funciones de regresión lineal es como si se anulara dicha suma y ocurriera el resumen de la red neuronal comentada.

Para solucionar este problema, a la salida de cada neurona se le aplica una función de activación que significa hacer una deformación no lineal a dicha salida, de manera que al sumar estas salidas no ocurra el cuello de botella. Existen numerosas funciones de activación, según que tipo de deformación se quiera aplicar a las salidas, algunos de los ejemplos más utilizados son la sigmoide, sinusoidal, gaussiana o ReLU, cada una con sus propios beneficios.

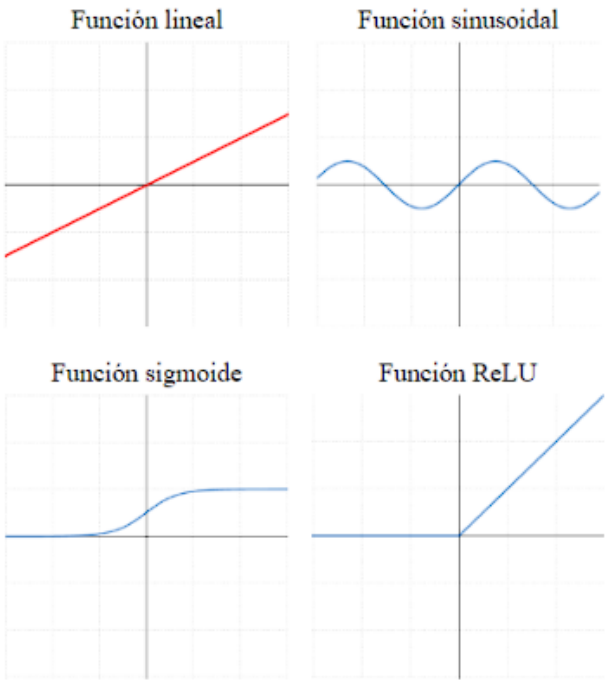

Figura 2.3: Función de regresión lineal y funciones de activación

2.3. Multilayer Perceptron

Deep Learning tiene diversos tipos de redes neuronales, según qué mecanismos utilizan para aprender de los errores o en qué dirección fluyen los datos, entre otras características.

Los modelos que se van a comparar en este proyecto parten del algoritmo supervisado Multilayer Perceptron. La red neuronal de este modelo es feed-forward por lo que las neuronas no tienen retroalimentación y los datos van en una sola dirección.

Para entrenar este tipo de redes se emplea el mecanismo de backpropagation, el cual comienza su aplicación al llegar los datos a la capa de salida. Cuando esto ocurre se obtendrán unos valores de comparar la salida esperada con la obtenida por la red y se calculará el error cuadrático medio que será transmitido a la neurona directamente anterior, desde la capa de salida hasta la de entrada y pasando por todas las neuronas de la red. En este proceso a cada neurona se le aplica el descenso de gradiente, es decir, se ajustan los parámetros usados para calcular la regresión lineal de dicho perceptrón con derivadas parciales y así saber cuanta 'culpa' tiene la neurona del error final. Cuando ya se conozca este valor de culpabilidad se propaga hacia atrás, de manera que al llegar a las neuronas de la capa de entrada se tendrá un vector con los valores de responsabilidad de cada neurona de la red, todo esto con una sola pasada a la red, por lo que es bastante eficiente.

Capítulo 3

Modelos del estudio

En este apartado se desarrolla el funcionamiento de los dos modelos que serán el caso de estudio de este trabajo. Para ello se explica de qué concepto parten y como se crea la arquitectura de cada modelo alrededor este.

3.1. Vision Transformer

La aparición de los mecanismos de atención junto a la arquitectura del Transformer en 2017 con el documento Attention Is All You Need [3] gracias a unos desarrolladores de Google, dieron un giro al campo del Deep Learning, concibiendo el concepto de 'atención'.

Aplicado a esa estructura, que será explicada próximamente, en el documento An Image Is Worth 16X16 Words: Transformers For Image Recognition At Scale del 2021 [4] se aplicó el modelo al campo de las imágenes, donde la estructura varía un poco de la de un Transformer, pero con el concepto de 'atención' como eje central.

3.1.1. Mecanismos de Atención

Este tiene como objetivo dar al modelo una capacidad que tenemos los seres humanos de prestar más o menos interés a alguna sección concreta de una imagen o a una palabra en una oración. Busca suplir la 'falta de memoria' de los modelos, es decir, solucionando el problema del peso de relación que tienen las primeras palabras de una frase muy larga con las últimas, el cual cuanto más larga fuera una frase, menos relación para el modelo iba a existir en los extremos de la oración.

Antes de comenzar con el cálculo de la atención se transforman los datos a vectores. En el caso de las palabras, cada una se pasará a un vector y en el de las imágenes estas se dividirán por partes o parches y estas piezas se traducirán en vectores.

El funcionamiento de la 'atención' comienza analizando el objeto completo de una vez, en una frase analiza todas las palabras en paralelo y en una imagen se divide esta en partes o parches y se analizan de la misma manera. Para la aplicación de la atención se pasan los inputs (cada palabra o parche) por tres redes neuronales con sus propios pesos, resultando en un nuevo vector por cada red, $query(q), key(k) y value(v)$.

Teniendo en cuenta que no se trata de un solo input, sino de n inputs, a continuación se crea una matriz por cada tipo de vector donde se unen los resultados de la operación anterior. Desembocando entonces tres matrices, una para las $Queries(Q)$ que serían las características de interés de la imagen, otra para las $Keys(K)$ como las características que podrían ser relevantes para Q y la última para los $Values(V)$ que son las características originales que se quieren escalar a probabilidades.

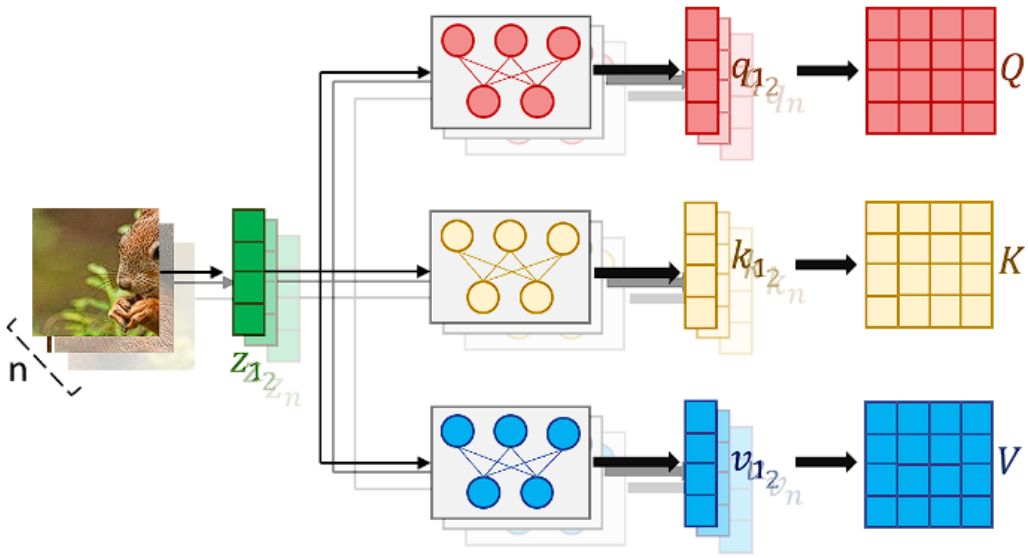

Figura 3.1: Creación de las matrices Q, K y V a partir de una imagen

Cuando ya se tengan estas tres matrices se va a calcular la Matriz de Atención mediante una función Atención(Q, K, V), que también es llamada Scaled-Dot Product Attention. Esta matriz final se calcula haciendo el producto entre Q y K traspuesta (QK^T), a continuación se escalará la matriz resultante dividiéndola entre la raíz de su dimensión $(\sqrt{d_K})$. Y se aplicará la función softmax para mostrar los valores de la matriz como probabilidades o importancia entre 0 y 1. Por último se hará el producto de la matriz de atención con V quedando como resultado de los pasos anteriores y teniendo ya las tres matrices, la fórmula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

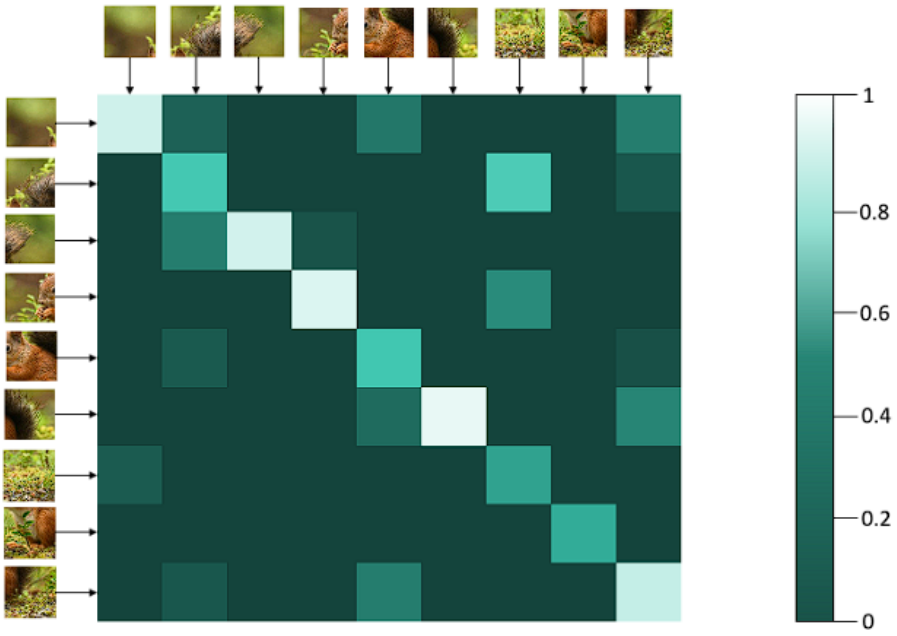

Figura 3.2: Matriz de atención con los parches de una imagen

3.1.2. Arquitectura

Conociendo el funcionamiento del mecanismo de atención se procederá a explicar la estructura del modelo Vision Transformer.

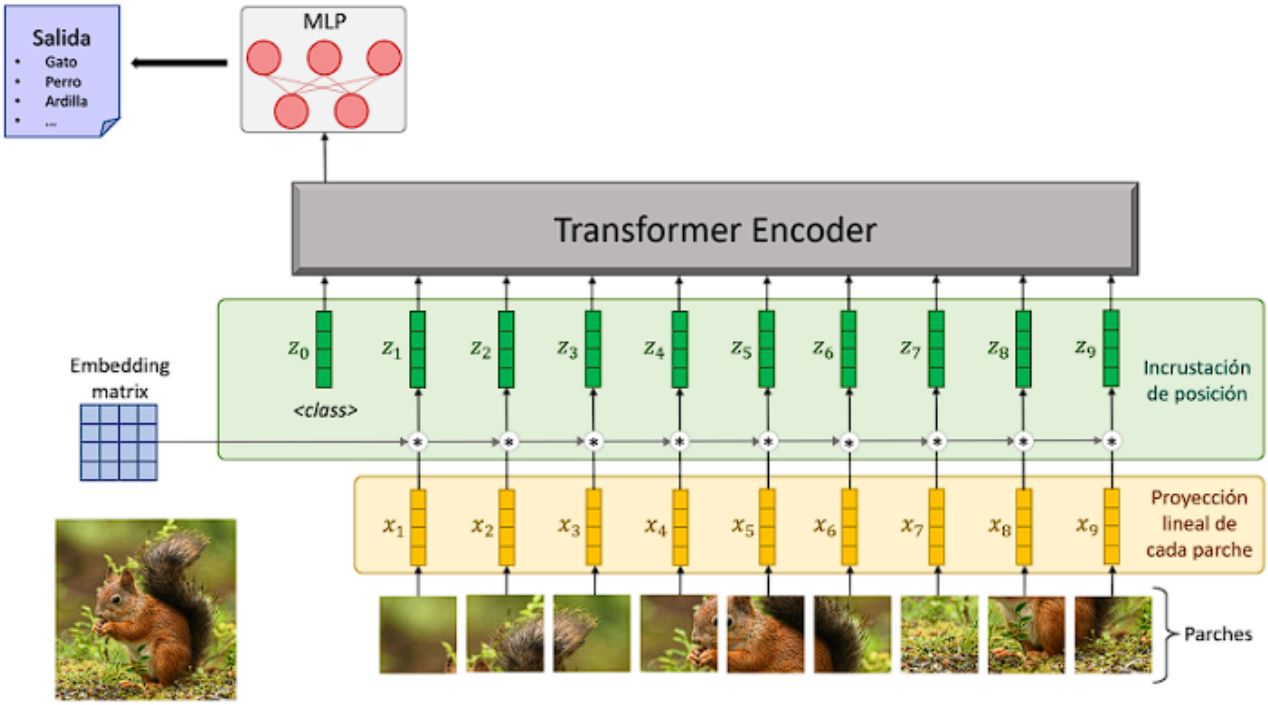

Figura 3.3: Arquitectura del modelo Vision Transformer

El modelo se puede dividir en tres partes, la primera es la preparación de la imagen donde esta se segmenta en parches que serán procesados en paralelo.

A continuación cada parche se proyectará como tokens unidimensionales (vectores) y finalmente se le sumará a cada token la posición del parche en cuestión según la fila que corresponda del Embedding Matrix, dado que los tokens serán tratados al mismo tiempo y no de manera secuencial. En esta primera parte de la arquitectura se crea un token adicional vacío (z_0) que tendrá las características generales de la imagen completa y será el que se utilice para clasificar la misma.

La segunda sección del modelo es el codificador del Transformer. La entrada de este bloque son los tokens previamente preparados, incluido el nuevo token vacío, donde serán procesados y la salida de este será el primer token, pero ya no estará vacío sino que tendrá las características de la imagen. En la próxima sección se profundizará en este bloque.

Por último, la tercera sección se compone de un Multilayer Perceptron (MLP) que tiene como entrada el token con las características de la imagen (z_0) para clasificarla y darle una de las etiquetas existentes.

3.1.3. Codificador Transformer

El codificador está compuesto por dos bloques mayores, ambos están compuestos primero por una capa de normalización y al final con una conexión residual cada uno. Cabe mencionar que ejecutan L codificadores idénticos secuencialmente, pero con pesos diferentes.

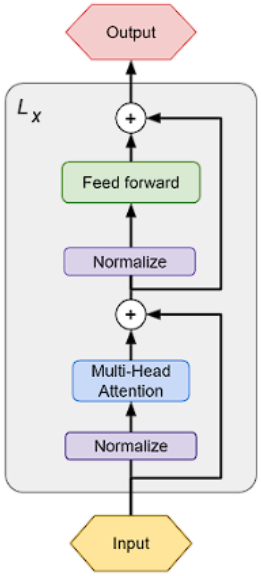

Figura 3.4: Bloque del codificador del Transformer

Normalize

Este primer bloque mejora el tiempo que tarda el modelo en ser entrenado y en general los resultados finales. Para eso las entradas, que son el conjunto de tokens z y estandariza cada uno utilizando su media y desviación estándar, dando como salida los tokens pero normalizados[5].

Multi-Head Attention

El input de este apartado son los tokens previamente normalizados y antes de aplicar el propio mecanismo del bloque se calculan las matrices Q, K y V como se explicó en apartado de atención.

Esta parte se resume en aplicar el algoritmo Scaled-Dot Product mencionado antes n veces y las salidas de este se concatenan para tener una sola y luego se les aplica proyección lineal para volver a tener los tokens en las mismas dimensiones que la entrada.

Este bloque es un símil de aplicar n filtros al input y teniendo en cuenta un parámetro W que es diferente para cada matriz y cada aplicación del Scaled-Dot Product, se podría representar la sección con las siguientes funciones:

$$Multi - HeadAttention = Linear(Concatenate(Scaled_1, Scaled_2, ..., Scaled_n))$$
$$Scaled_n = Attention(QW_n^Q, KW_n^K, VW_n^V)$$

Conexión Residual (Add)

La función de este apartado es mejorar los resultados del entrenamiento del modelo, enfocándose en solventar dentro de lo posible el ruido que se genera al utilizar la salida de un bloque como la entrada de otro continuamente. Para eso se toma la salida de haber ejecutado un conjunto de bloques secuencialmente y se le suma la entrada del primer bloque.

En este caso se sumaría la salida del bloque Multi-Head Attention con los datos antes de ser procesados por el bloque de normalización.

Feed-Fordward

Este bloque no es más que un MLP previo a la clasificación de las imágenes por sus características. Sirve para adaptar los tokens que recibe para tener mejores resultados al clasificar la imagen en el MLP situado fuera del codificador.

Funcionamiento en conjunto

Primero al codificador entran todos los tokens de los que se hará una copia C que se empleará más adelante. Para la primera agrupación los tokens son pasados por la capa de normalización, para inmediatamente pasar el resultado de esta capa por el Multi-Head attention que crea las tres matrices a partir de su entrada y devuelve un conjunto de nuevos tokens. A la salida del Multi-Head Attention se le hace una conexión residual con la copia generada antes C y el resultado de esta suma pasa a la segunda agrupación.

En esta segunda parte se vuelve a hacer una copia de los tokens de las operaciones anteriores C', se pasan los vectores por una capa de normalización y seguidamente se meten a un MLP. Con la salida de esta red neuronal se hace una conexión residual con C' y ya se tendrían las salidas que serán la misma cantidad de tokens que la entrada del codificador, en el cual el primer token ya no estará vacío y tendrá las características que correspondan.

3.2. Gated Multilayer Perceptron

Con el objetivo de investigar la necesidad de los mecanismos de atención en los Transformer se desarrolla un modelo que es descrito en el documento “Pay Attention to MLPs”[6] en 2021 por un grupo de desarrolladores de Google.

Este modelo se basa en el antes nombrado Multilayer Perceptron junto al funcionamiento de Transformer, pero sin hacer uso de los mecanismos de atención.

3.2.1. Mecanismo de Gating

Este concepto [7] viene acompañado de los pesos que utilizan las neuronas en cada momento, guardando algunos de esos pesos ya utilizados y los nuevos que se generen. Con el gating al emplear una neurona, esta tendrá un conjunto de vectores de peso, de los cuales uno solo estará activo y será el que se aplique junto al input dentro de la neurona.

3.2.2. Arquitectura

La arquitectura de este modelo es muy similar a la del Vision Transformer pero quitando algunas funciones y por supuesto con el componente principal (El bloque del propio modelo) diferente.

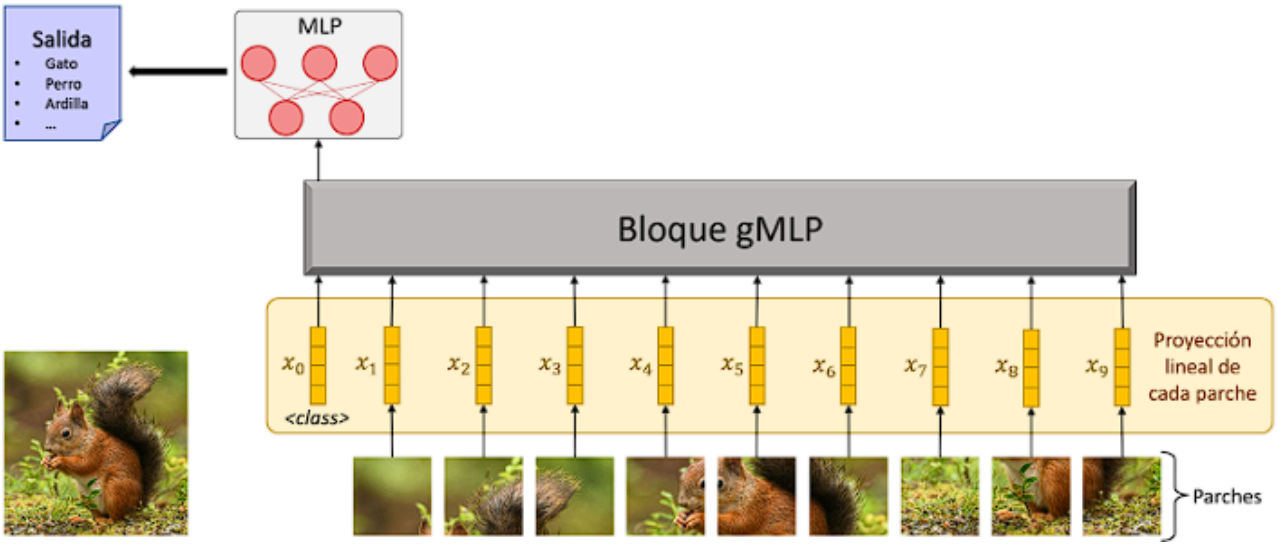

Figura 3.5: Arquitectura del model Gated Multilayer Perceptron

Igual que el primer modelo, este se puede dividir en tres secciones. La primera es la preparación de las imágenes, donde se divide esta en parches y se proyecta cada uno para obtener tokens o vectores; Se crea un token adicional vacío que tendrá las características finales de la imagen y que será el empleado en la última parte del modelo. Como se puede observar en esta arquitectura no se añade la posición de cada parche dado que esta información espacial será buscada de otro modo dentro del bloque gMLP. La segunda sección, que se desglosará en el siguiente sub apartado, es el bloque gMLP que procesará la imagen y obtendrá las cualidades de la imagen que serán procesadas en la última sección, una red Multilayer Perceptron que etiquetará la imagen dándole como entrada el token x_0 que contendrá lo necesario para poder clasificar la imagen.

3.2.3. Bloque gMLP

Al estar en parte basado en el codificador del Transformer, hay partes que explicadas en la sección anterior que se utilizan aquí también, por lo que se omitirán dichas partes. Estas son: la normalización, la función de activación y la conexión residual.

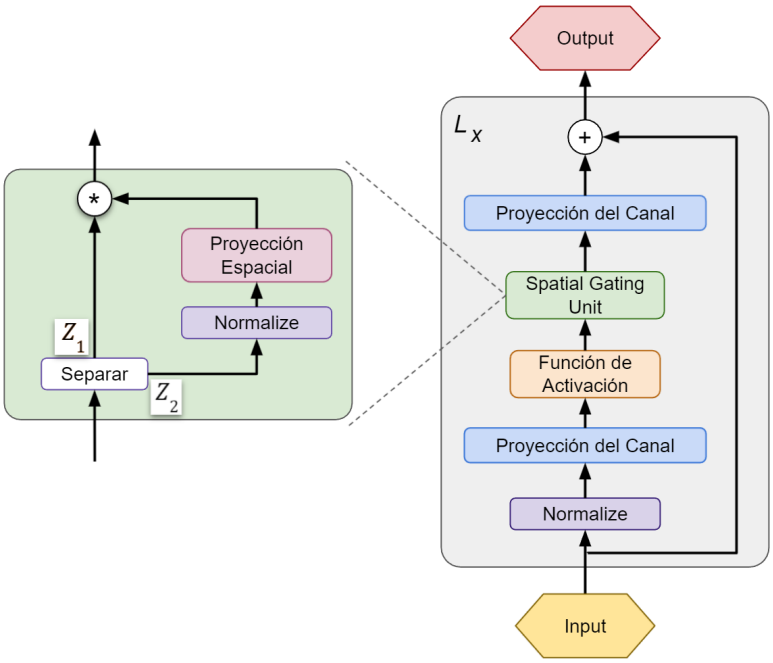

Figura 3.6: Bloque de gMLP

Proyección de canal

Proyecta los tokens a lo largo de los canales que conforman la imagen en sí para preparar y simplificar el conjunto de tokens.

Función de activación

Aplica las deformaciones no lineales necesarias para evitar el cuello de botella que se puede generar al ejecutar una red neuronal únicamente con operaciones lineales. En este caso se hace uso de GeLU (Gaussian Error Linear Unit).

Spatial Gating Unit

Esta parte es la que sustituye a la aplicación de la atención en los Transformers y tiene la finalidad de habilitar la interacción entre tokens. Su entrada es un conjunto de tokens o matriz Z que se va a dividir en dos partes Z_1, que no va a ser procesada, y Z_2 que será empleada en dos operaciones.

Aquí Z_2 primero se pasa por una capa de normalización y luego se le aplicará proyección lineal con una matriz W generada independientemente de la entrada del bloque. Esta matriz irá variando sus valores para aprender, a lo largo de las ejecuciones, como deben interactuar los tokens entre sí, aunque sus valores iniciales son lo más cercano a 0 posibles, de manera que las interacciones entre los tokens sean casi nulas. El último dato que se usa en esta operación es el bias b, quedando la función hasta el momento: $Z_2' = WZ_2 + b$

Con esta nueva matriz Z_2' se hará el producto de Hadamard $(\odot)$, es decir, multiplicar las matrices Z_1 y Z_2, resultando finalmente SGU completo en la siguiente función:

$$SGU(Z) = Z_1 \odot (WZ_2 + b)$$

Funcionamiento en conjunto

La manera en que funciona el Bloque gMLP comienza haciendo una copia C de los tokens de entrada para utilizarla al final en una conexión residual. Se pasan la matriz de tokens por una capa de normalización, se proyectan a lo largo del canal y se pasa el resultado por una función de activación.

El siguiente paso es dar como entrada la matriz resultante de las operaciones anteriores a la sección SGU, que conseguirá una nueva matriz de la interacción entre los tokens, para hacer una última proyección sobre el canal y aplicar una conexión residual con la copia C hecha al comienzo del bloque. Todo este procedimiento se repite tantas veces como bloques gMLP haya.

Capítulo 4

Datasets e Implementación

Tanto la implementación de los modelos como la preparación necesaria de los datasets están hechos con Python 3.10.4 y con la biblioteca especial para Machine Learning Tensorflow y de este se emplearon los métodos para las capas de los modelos de Keras.

4.1. Datasets

Por lo general, los datasets usados son los que se proveen de Keras mediante load_data() que devuelve cuatro parámetros, dos son para el entrenamiento, correspondiendo a vectores NumPy con las imágenes y sus etiquetas. Y los otros dos parámetros es lo mismo pero con los datos de pruebas.

4.1.1. MNIST

El conjunto de imágenes más sencillo es el de los números escritos al alza[8]. Se compone de imágenes que van desde el cero al nueve y con diferentes fuentes para cada dígito. De sus 70.000 ejemplos, 60.000 son para el entrenamiento y 10.000 para las pruebas. Todas las fotos están en escala de grises y tienen tamaño 28x28. Pueden cargarse sin problema desde los dataset que ofrece Keras.

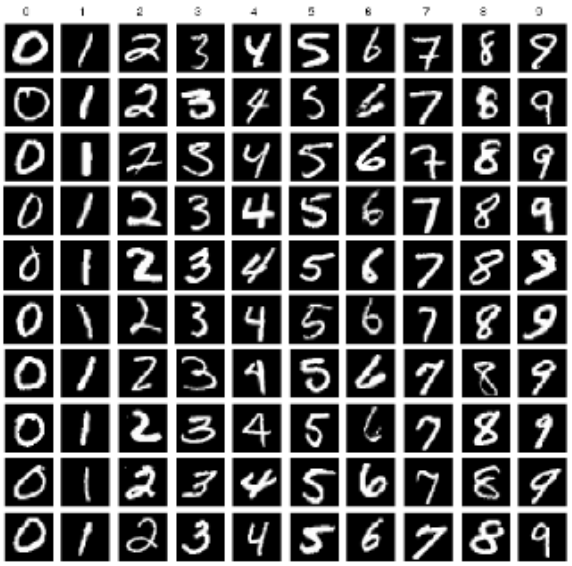

Figura 4.1: Ejemplos de imágenes de MNIST

4.1.2. CIFAR-10

Este dataset tiene conceptos más complejos y utiliza un rango de colores más amplio que el caso anterior[9]. Se compone de diez clases de imágenes compuestas por vehículos y animales: Aviones, coches, pájaros, gatos, ciervos, perros, ranas, caballos, barcos y camiones. Esta base de datos se compone de un total de 60.000 imágenes, de las cuales 50.000 están destinadas al entrenamiento y 10.000 a las pruebas posteriores, todas ellas con un tamaño de 32x32 y se puede emplear con las funcionalidades de Keras.

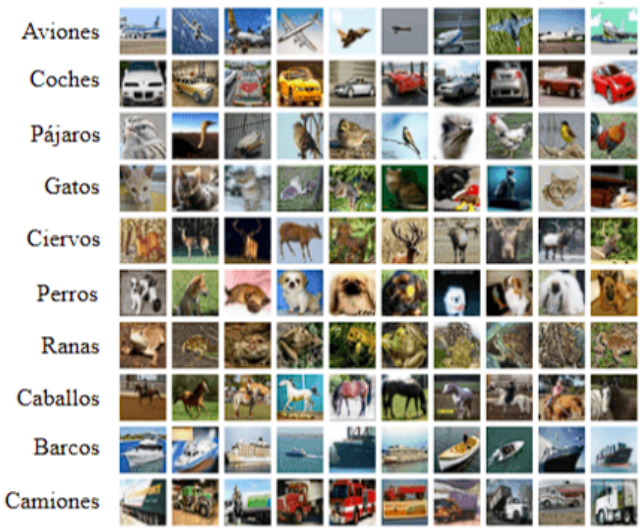

Figura 4.2: Ejemplos de imágenes de CIFAR-10

4.1.3. CIFAR-100

Es el dataset más complejo por su cantidad de clases, es así porque se compone de cien clases que se agrupan en veinte superclases, como por ejemplo flores, insectos, muebles, dispositivos electrónicos o frutas y vegetales, entre muchos otros[9]. Todas las imágenes se forman a partir del espectro de colores normal (rojo, azul y verde) y tienen tamaño 32x32. De cada clase hay 500 imágenes que son para el entrenamiento y otras 100 para los tests, por lo tanto, en total hay 50.000 fotos para entrenar los modelos y otras 10.000 para las pruebas. Se puede utilizar este conjunto de datos con los métodos de Keras.

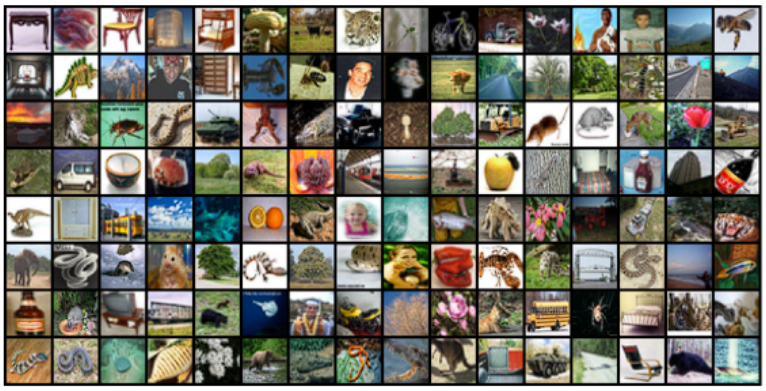

Figura 4.3: Ejemplos de imágenes de CIFAR-100

4.1.4. FER-2013

Otro dataset de los más complejos es FER-2013. Está compuesto por imágenes de tamaño 48x48 en escala de grises y son diferentes caras que se dividen por clases según el sentimiento que muestren. Losa sentimientos se dividen en siete categorías que son: Enfado, asco, miedo, felicidad, neutralidad, tristeza y sorpresa, además hay un total de 32.298 fotos que se segmentan en 28.709 para entrenar y 3.589 para los tests[10].

Este dataset no está entre los que dispone Keras, por lo que hay que obtenerlo mediante otros métodos. Primero es necesario descargar el documento .csv que tiene estas imágenes descritas mediante tres columnas: Clase de la imagen, Píxeles que componen la imagen como un string y Uso de la imagen (entrenamiento o pruebas). El siguiente paso es dividir las imágenes (en strings) según sean de entrenamiento o pruebas, para luego transformar el string que describe una imagen en una lista y por último redimensionarla a los 48x48 píxeles que conforman la foto, dando por fin a la imagen en sí. Para terminar hay que guardar y enlistar las etiquetas o clases del dataset para usarlas en los modelos junto a las imágenes.

Figura 4.4: Ejemplos de imágenes de FER-2013

4.2. Modelos

Para la implementación de los modelos se hizo uso de la librería antes mencionada Keras en Python partiendo de las implementaciones preexistentes de esta librería, aunque con algunas funciones adicionales.

4.2.1. Compilación de los modelos

La compilación de los dos modelos es bastante similar y es importante destacar que mecanismos de optimización y la obtención de los valores de pérdida que se quieren minimizar en todo momento.

Optimizador

Para mejorar los resultados con cada época, modificando los pesos y reduciendo el error, se utilizan los optimizadores. Los dos modelos se sirven de AdamW[11] que es el optimizador Adam[12] pero con decadencia de pesos (weight decay). Este calcula el descenso de gradiente, se apoya en la penalización del decaimiento de pesos y tiene en cuenta los gradientes anteriores para saber como modificar cada peso.

Función de pérdida

Para conocer qué tan buenos son los resultados que se están obteniendo en un modelo para apoyar al optimizador a conocer hacia donde modificar tanto los pesos como los sesgos de la neurona, se emplea una “Loss function” o función de pérdida.

Hay diversas maneras de calcular estos valores y en la implementación de ambos modelos se utiliza la Entropía Cruzada o Cross Entropy que es el estándar de cálculo de pérdida más utilizado y se puede representar con la siguiente función:

$$Perdida = -\sum_{i=0}^{n^o clases} y_i \ln(\widehat{y})$$

Esta función lo que significa es con (y) que es una matriz de identidad de las clases, se seleccione a qué clase pertenece la imagen realmente (clase correcta) y se hace el logaritmo neperiano del valor que obtuvo el modelo para la clase correcta en esa imagen (clase predicha).

Cuanto mayor sea el valor, más alejadas están las predicciones en línea general de los resultados reales, pudiendo verse en una representación con un punto de ejemplo. Teniendo que una imagen realmente es un perro y la predicción dice que esa imagen es un 0.9, es decir, indica que es un 90 % perro y, por lo tanto, se clasifica como perro, su pérdida dado que ha acertado en la predicción. En consecuencia, tendríamos que la pérdida de esta predicción es $-ln(0{,}9) = 0{,}105$. De manera que aplicando este método a todas las predicciones a lo largo de la ejecución del modelo se pueda obtener una gráfica representativa.

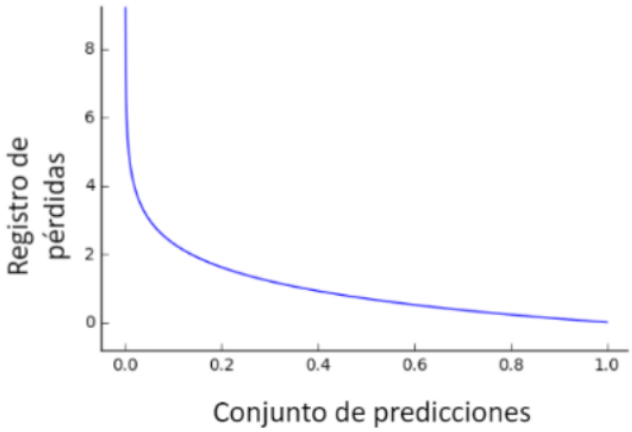

Figura 4.5: Encima el registro de valor de pérdidas a lo largo del modelo

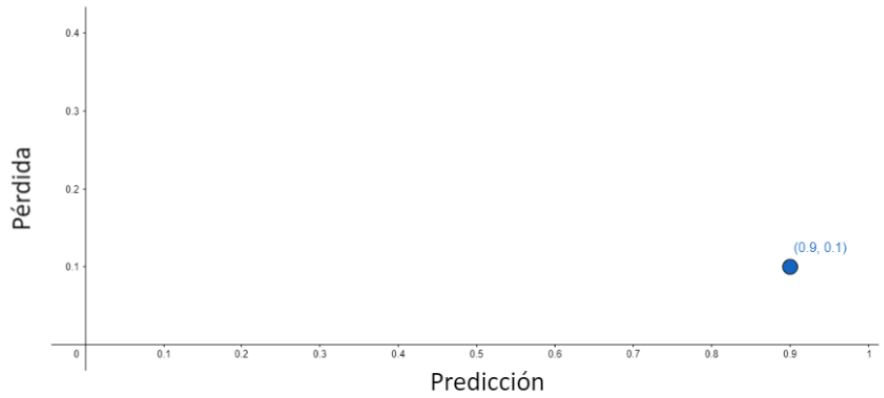

Figura 4.6: Debajo representación del valor de pérdida para el ejemplo anterior

4.2.2. Funciones añadidas

En cuanto a las mejoras o diferencias, al ser necesaria la ejecución continua para encontrar la mejor combinación de hiperparámetros, se utilizan ficheros con los valores con el conjunto de valores con que se quiere ejecutar el modelo, donde cada línea de ese modelo pertenece a una ejecución. Además, para evitar que el usuario esté pendiente del final de cada ejecución para ver los resultados, estos se almacenan en unos documentos de registro junto a los parámetros usados para que den esos resultados, los propios resultados y la fecha en que se ejecutó esa línea. Adicional a esta parte, para la creación de gráficas sencillas se hizo un programa adicional para formatear los registros y ordenarlos.

Otra funcionalidad relacionada con el muestreo de los datos, es el guardado de los modelos en formato SavedModel donde se almacenan los modelos compilados y entrenados y que luego son cargados en otro programa para obtener predicciones y gráfica sobre estas.

Para acelerar las obtenciones de resultados y con una GPU disponible, se habilitó el uso de esta mediante Tensorflow, de manera que se reduzcan mucho los tiempos de ejecución de ambos modelos y se puedan obtener más datos en intervalos menores. Además, para disminuir más el tiempo no solo de procesado, sino la cantidad de veces que es necesario entrenar los modelos para obtener mejores resultados, se añadió la prevención de sobreentrenamiento con una paciencia generalmente entre 10 y 50 épocas; Monitorizando los cambios de la validación de pérdidas y en el caso de que se accione el mecanismo de prevención de sobreentrenamiento, se guarden los pesos del mejor entre las últimas épocas en el rango que tenga la paciencia.

Capítulo 5

Comparativa

Los resultados de los cuatro datasets para cada modelo fueron guardados y tratados con programas en Python mencionados en el capítulo anterior. Aquí se verán los resultados obtenidos según qué parámetros y se van a comparar los crecimientos o decrecimientos de estos entre los dos modelos.

5.1. Resultados de gMLP

5.1.1. Dataset CIFAR-10

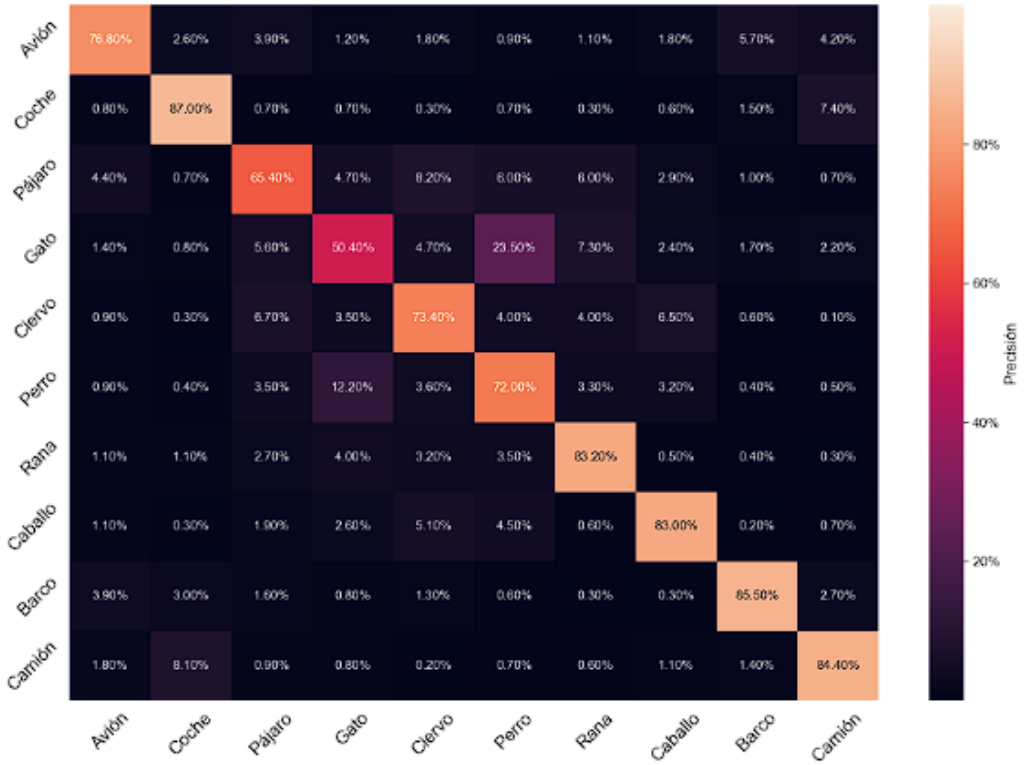

Figura 5.1: Matriz de confusión para CIFAR-10 con gMLP

Para visualizar estos resultados, tenemos una matriz de confusión que se asemeja mucho a una de identidad, por lo que los resultados son muy buenos, destacando que al modelo se le dificulta ligeramente diferenciar entre perros y gatos. Puede ser debido a

que sus características son muy similares, comparándolo con un vehículo u otro animal de los presentes en el dataset.

Como se puede ver en la tabla (Tabla 5.1), las épocas reales realizadas nunca alcanzan las establecidas, existiendo una mejoría de la precisión del modelo cada vez que aumentaban las épocas hechas, pero con un límite cercano a 150, dando a entender que si se sobrepasa ese límite cerca o por encima de las épocas solicitadas los resultados empeorarían Los hiperparámetros por sí solos no aparentan tener relación con la mejora del modelo, pero observando en conjunto la cantidad de épocas, acompañada de una dimensión embedding no superior a 256, con no más de 8 bloques y para ajustar con un weight decay de valores bajos, de 0.001, mejoran el rendimiento. Añadido a esto, si se redimensiona la imagen al doble, 64x64 o se deja en su tamaño original, junto a parches de 8x8 se obtienen los mejores resultados.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	128	200	0,2	64	8	64	256	8	192	130	78,66 %	98,88 %
0,0001	128	200	0,2	64	8	64	128	8	192	86	78,35 %	98,88 %
0,0001	128	200	0,2	64	8	64	128	4	192	72	77,64 %	98,57 %
0,0005	256	200	0,2	64	8	64	256	4	192	57	77,52 %	98,57 %
0,0010	512	200	0,2	64	8	64	256	4	192	61	77,27 %	98,50 %
0,0005	512	200	0,2	64	8	64	256	4	192	59	77,20 %	98,45 %
0,0001	128	200	0,2	64	8	64	256	4	192	64	77,12 %	98,55 %
0,0001	128	300	0,2	64	8	64	256	8	192	102	76,81 %	98,59 %
0,0005	512	200	0,2	64	12	25	256	4	432	86	75,11 %	98,17 %
0,0005	256	300	0,1	32	8	16	256	8	192	88	75,05 %	97,91 %
0,0001	128	200	0,2	32	8	16	256	8	192	105	74,79 %	98,20 %
0,0005	128	200	0,2	64	8	64	256	4	192	48	74,64 %	98,41 %
0,0001	128	200	0,2	32	8	16	128	8	192	81	74,43 %	98,08 %
0,0005	256	200	0,2	64	12	25	256	4	432	52	74,18 %	98,32 %
0,0001	128	300	0,2	64	8	64	512	8	192	121	74,01 %	98,12 %

Tabla 5.1: Configuración de hiperparámetros con los mejores resultados de gMLP con CIFAR-10

5.1.2. Dataset MNIST

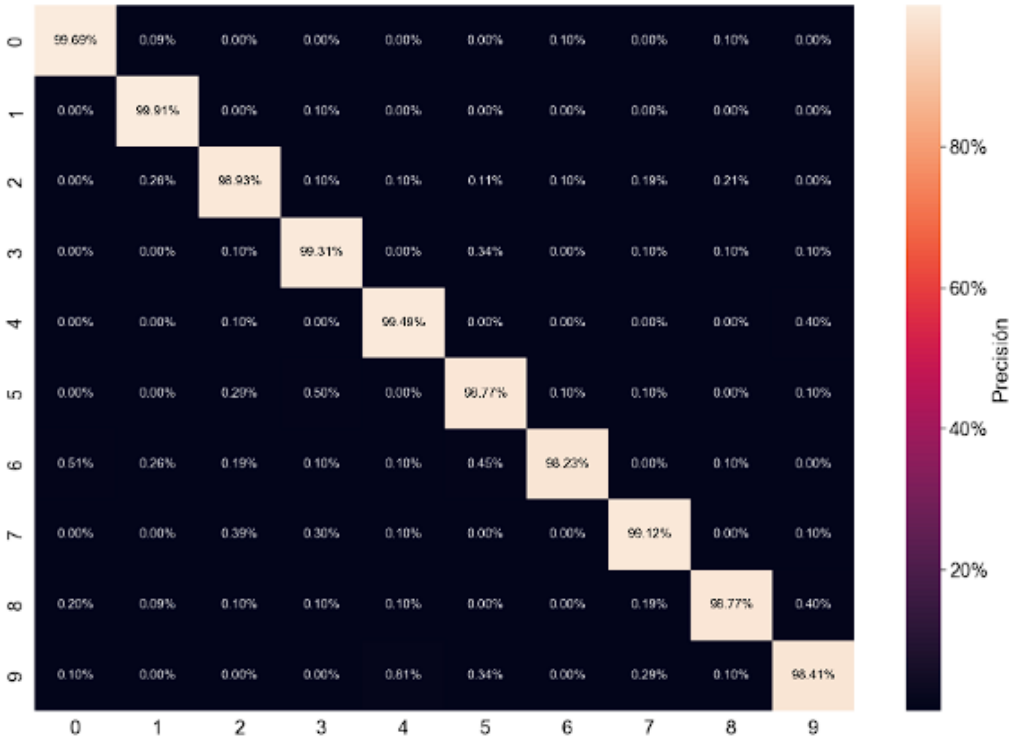

Figura 5.2: Matriz de confusión para MNIST con gMLP

De manera similar, pero con mejores resultados que con el dataset anterior, con MNIST el modelo parece diferenciar bien los valores sin tener algo destacable que remarcar.

Para este dataset (Tabla 5.2) lo más efectivo fue aumentar ligeramente el tamaño de las imágenes y utilizar un tamaño de parche no superior a 8 y tener pocos bloques, ya que no se requiere de gran cantidad de Bloques gMLP y una decadencia de pesos baja también. Como se observa, al tratarse de un dataset bien diferenciado no requiere de muchas épocas en comparación al resto para tener resultados excelentes.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	128	100	0,2	32	8	16	256	4	192	75	99,15 %	100,00 %
0,0001	128	100	0,2	64	8	64	256	4	192	60	99,14 %	99,99 %
0,0005	512	100	0,2	64	8	64	256	4	192	61	98,93 %	100,00 %
0,0001	128	100	0,2	32	8	16	256	8	192	80	98,92 %	100,00 %
0,0005	128	100	0,2	64	12	25	256	4	432	49	98,89 %	99,99 %
0,0005	256	100	0,2	64	8	64	256	4	192	40	98,86 %	99,99 %
0,0001	128	100	0,2	32	8	16	128	4	192	71	98,85 %	100,00 %
0,0001	128	100	0,2	72	8	81	256	8	192	53	98,82 %	99,98 %
0,0001	128	100	0,2	64	8	64	256	8	192	70	98,79 %	99,97 %
0,0001	128	100	0,2	72	8	81	256	4	192	51	98,74 %	100,00 %
0,0005	512	100	0,2	64	12	25	256	4	432	47	98,74 %	100,00 %
0,001	512	100	0,2	64	12	25	256	4	432	48	98,72 %	99,99 %
0,0005	128	100	0,2	64	8	64	256	4	192	39	98,68 %	99,99 %
0,0001	128	50	0,2	64	8	64	256	8	192	50	98,66 %	100,00 %
0,0005	256	100	0,2	64	12	25	256	4	432	35	98,61 %	99,97 %

Tabla 5.2: Configuración de hiperparámetros con los mejores resultados de gMLP con MNIST

5.1.3. Dataset CIFAR-100

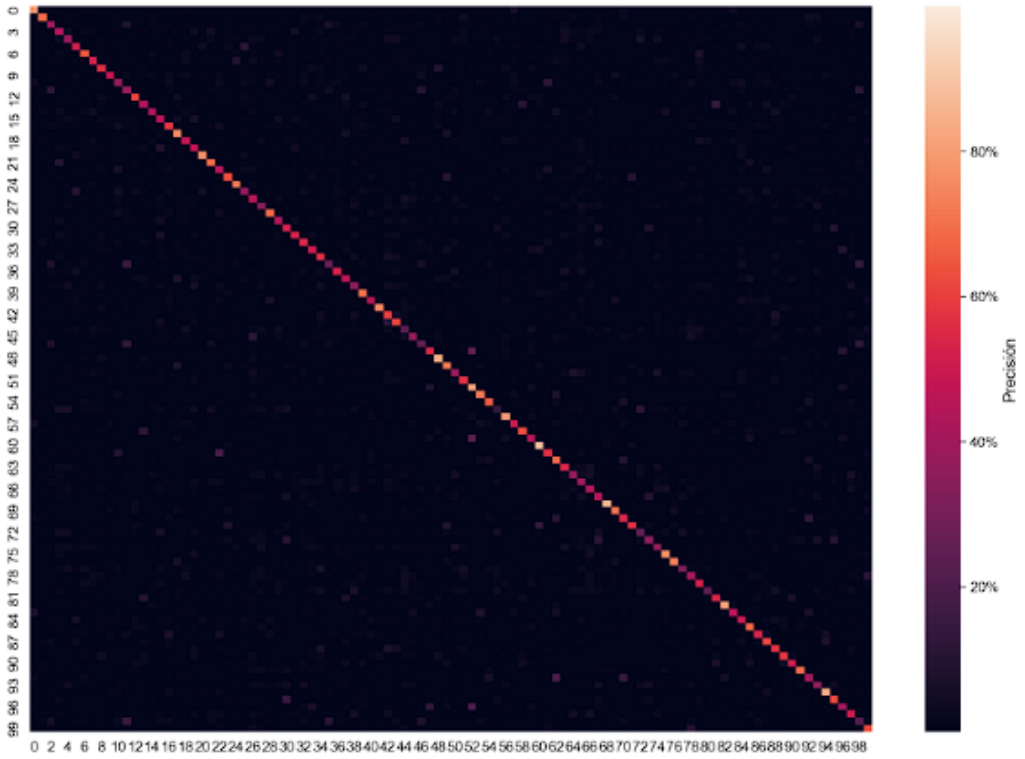

Figura 5.3: Matriz de confusión para CIFAR-100 con gMLP

Se puede observar que dentro de lo que cabe esta matriz se asemeja bastante a una de identidad. Pero a pesar de parecer que tiene buenos resultados, estos rondan entre el 50 % y 60 %, se debe a que se pueden ver ciertas clases que se confunden con otras, aunque al estar dispersas no parecen ser muchas.

Este modelo (Tabla 5.3) no mejora los resultados con más épocas, pero sí con más bloques y más unidades ocultas, aunque con cierto límite, donde al superar los 256 ya no se obtienen valores tan buenos. Además, con tamaños de Batch inferiores y un aumentado de imagen no superior a 64 se ven resultados mejores, aunque sin alcanzar unos realmente buenos.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	128	100	0,2	64	8	64	128	8	192	62	53,62 %	81,31 %
0,0001	128	200	0,2	64	8	64	128	8	192	68	53,26 %	81,37 %
0,0005	512	300	0,2	64	8	64	128	8	192	98	52,94 %	81,31 %
0,0001	128	200	0,2	64	8	64	256	6	192	68	52,83 %	80,54 %
0,0008	1024	300	0,2	64	8	64	128	8	192	98	52,76 %	80,55 %
5,00E-05	512	300	0,1	64	8	64	64	12	192	177	52,37 %	80,62 %
0,0001	128	100	0,2	64	9	49	128	8	243	70	52,23 %	80,08 %
0,0001	128	200	0,2	64	9	49	128	6	243	67	52,10 %	80,37 %
8,00E-05	512	300	0,2	64	8	64	128	8	192	156	51,95 %	80,56 %
0,0001	128	200	0,2	64	8	64	128	6	192	62	51,94 %	80,39 %
0,0001	128	100	0,2	64	8	64	128	4	192	71	51,77 %	79,61 %
1,00E-05	512	300	0,2	72	8	81	64	8	192	300	51,73 %	80,66 %
0,0001	128	200	0,2	64	9	49	128	10	243	83	51,64 %	79,87 %
0,0001	1500	300	0,2	64	8	64	128	8	192	201	51,63 %	80,25 %
1,00E-05	128	200	0,2	64	8	64	64	8	192	78	51,52 %	80,80 %

Tabla 5.3: Configuración de hiperparámetros con los mejores resultados de gMLP con CIFAR-100

5.1.4. Dataset FER-2013

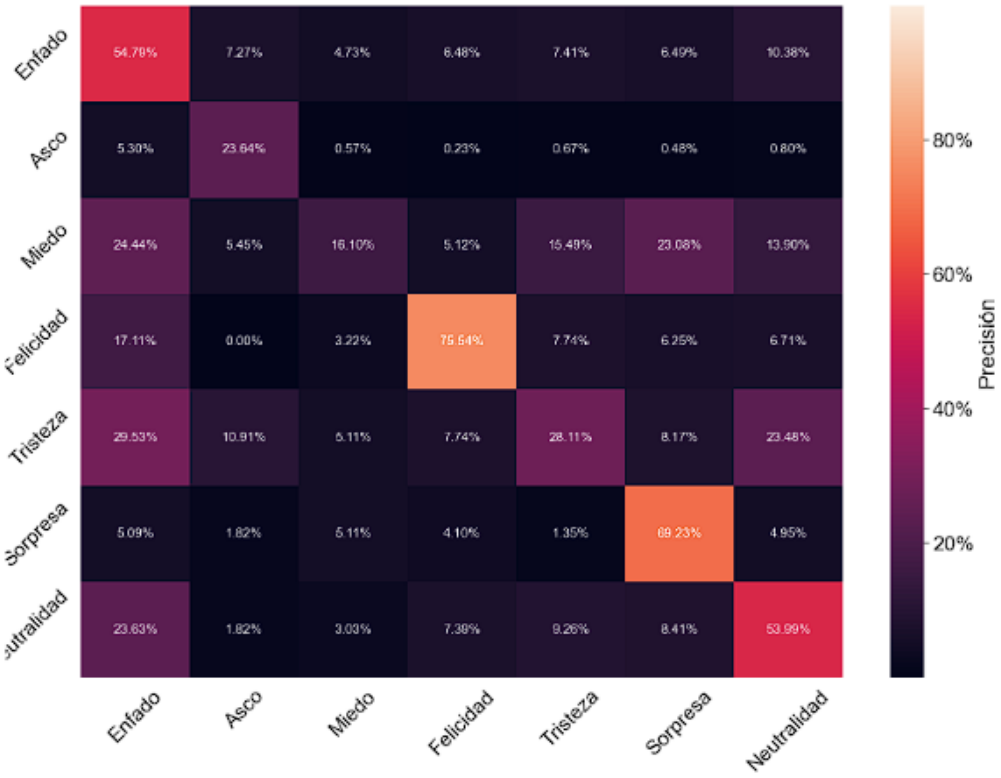

Figura 5.4: Matriz de confusión para FER-2013 con gMLP

Este dataset al que se interpreta como el más complejo, tiene una precisión relativamente baja, no alcanzando el 60 % con gMLP, es por ello que la matriz de confusión no es tan parecida a una de identidad. Por la figura se puede remarcar para el modelo la expresión de miedo, la más compleja de clasificar, errando el modelo clasificando el miedo más como enfado o sorpresa. Por otro lado, parece que para el modelo el enfado tiene bastante composición del resto de sentimientos, aunque sí se diferencia más que el miedo.

En este caso (Tabla 5.4), la mejoría se refleja en una decadencia de pesos mayor sin superar los 0,001, junto al tamaño del batch y las imágenes. Los parches menores de 8x8 empeoran los resultados y más de 4 bloques empeoran la precisión.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0005	256	300	0,2	64	8	64	256	4	192	93	55,34 %	98,11 %
0,0005	512	300	0,2	64	8	64	256	4	192	106	54,86 %	97,83 %
0,0001	128	300	0,2	72	8	81	256	4	192	131	54,25 %	97,77 %
0,001	256	300	0,2	64	8	64	256	4	192	98	54,25 %	97,72 %
0,001	512	300	0,2	64	8	64	256	4	192	99	54,22 %	97,55 %
0,0005	256	300	0,2	64	12	25	256	4	432	123	54,00 %	97,41 %
0,0005	512	300	0,2	64	12	25	256	4	432	120	52,94 %	97,69 %
0,0001	128	300	0,2	64	8	64	256	4	192	110	52,80 %	97,10 %
0,0001	128	300	0,2	32	8	16	128	4	192	126	52,47 %	96,91 %
0,001	512	300	0,2	64	12	25	256	4	432	101	51,99 %	97,24 %
0,0001	128	300	0,2	32	8	16	256	4	192	139	51,96 %	97,35 %
0,0005	128	300	0,2	64	8	64	256	4	192	85	51,77 %	97,41 %
0,0005	128	300	0,2	64	12	25	256	4	432	79	49,26 %	96,99 %
0,001	128	300	0,2	64	8	64	256	4	192	72	48,84 %	97,07 %
0,001	256	300	0,2	64	12	25	256	4	432	73	48,43 %	96,96 %

Tabla 5.4: Configuración de hiperparámetros con los mejores resultados de gMLP con FER-2013

5.2. Resultados de ViT

5.2.1. Dataset CIFAR-10

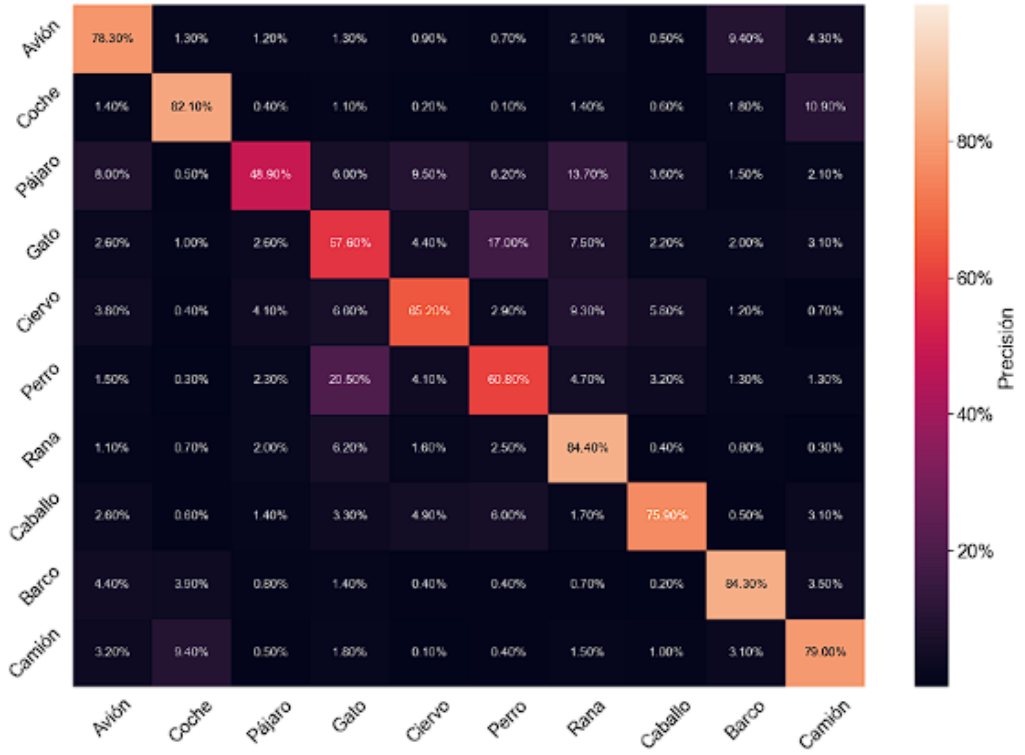

Figura 5.5: Matriz de confusión para CIFAR-10 con ViT

Este primer dataset tiene resultados bastante similares que para gMLP, pero en este caso el modelo nota menos las diferencias entre los gatos y los perros. Adicional a ello, los aciertos en las clases son menores en los animales que en los vehículos, no en gran medida, pero sí es notable si se compara con gMLP.

El aumento de las imágenes beneficia a los resultados (Tabla 5.5), así como mantener los parches a 8x8 y proyectar estos a vectores de mayores tamaños. Con más codificadores del Transformer y en líneas generales con más de 100 épocas, pero menas de 200 suelen ajustarse los resultados mejorándolos.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	128	300	64	8	64	72	8	12	192	152	83,41 %	99,03 %
0,0001	128	300	72	8	81	72	8	6	192	203	83,40 %	99,26 %
0,0001	128	300	72	8	81	72	8	6	192	177	83,32 %	99,13 %
0,0001	128	300	72	8	81	72	8	12	192	171	83,12 %	99,21 %
0,0001	128	300	72	8	81	72	8	12	192	125	83,05 %	99,12 %
0,0001	128	300	64	8	64	72	8	6	192	122	83,00 %	99,16 %
0,0001	128	300	64	8	64	72	8	12	192	161	82,64 %	98,98 %
0,0001	128	300	72	8	81	32	8	12	192	126	82,63 %	99,04 %
0,0001	128	300	72	8	81	32	8	6	192	154	82,46 %	99,05 %
0,0001	128	300	64	8	64	32	8	6	192	158	82,21 %	99,05 %
0,0001	128	300	72	8	81	32	8	12	192	148	82,17 %	99,33 %
0,0001	128	300	64	8	64	32	8	12	192	160	82,14 %	99,16 %
0,0001	128	300	64	8	64	72	8	6	192	168	82,02 %	99,07 %
0,0001	128	300	64	8	64	32	8	12	192	190	81,86 %	99,22 %
0,0001	128	300	72	8	81	32	8	6	192	113	81,84 %	99,12 %

Tabla 5.5: Configuración de hiperparámetros con los mejores resultados de ViT con CIFAR-10

5.2.2. Dataset MNIST

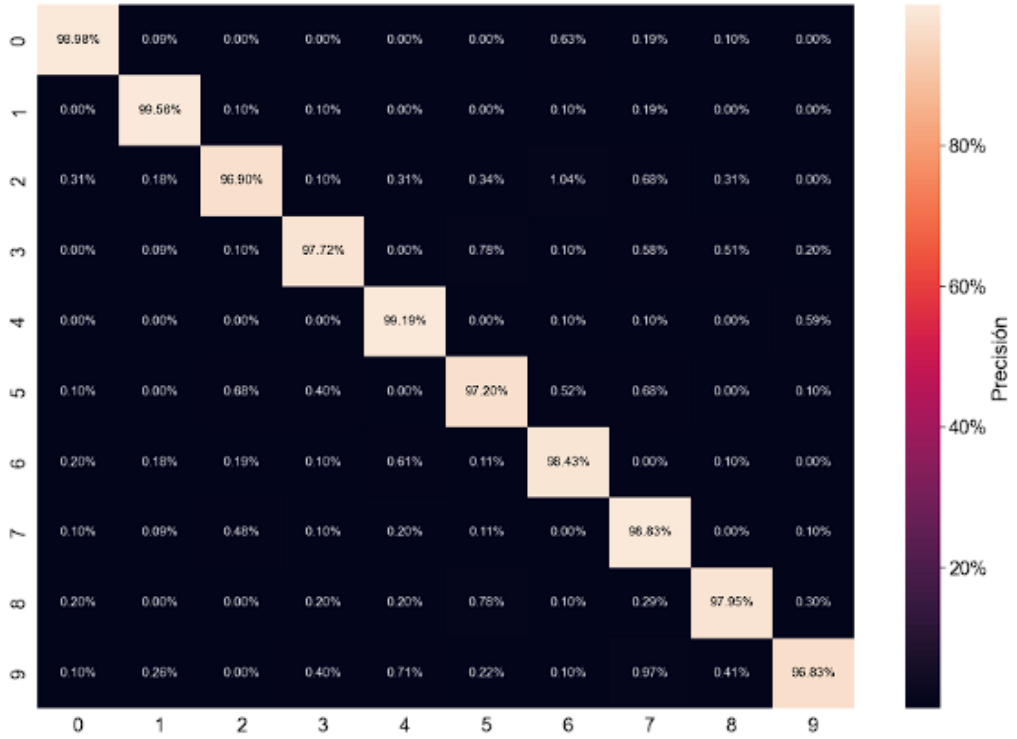

Figura 5.6: Matriz de confusión para MNIST con ViT

Al tratarse del dataset más sencillo, con un solo canal de color y con formas bien diferenciadas, prácticamente no hay error en las predicciones.

El dataset (Tabla 5.6) no requiere de tantas épocas para obtener buenos resultados, las imágenes a mayor escala si rebosar los 72 píxeles y con parches cuadrados de 8 píxeles dan resultados bastante buenos, sin ser necesario poner gran cantidad de codificadores secuenciales para los mejores resultados.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	128	300	72	8	81	72	8	6	192	99	99,02 %	99,99 %
0,0001	128	300	64	8	64	72	8	6	192	128	98,96 %	100,00 %
0,0001	128	300	64	8	64	72	8	12	192	132	98,93 %	99,97 %
0,0001	128	300	72	8	81	72	8	6	192	114	98,86 %	99,98 %
0,0001	128	300	72	8	81	72	8	12	192	106	98,82 %	99,97 %
0,0001	128	300	64	8	64	72	8	6	192	137	98,79 %	99,99 %
0,0001	128	300	72	8	81	32	8	6	192	155	98,67 %	99,99 %
0,0001	128	300	72	8	81	32	8	12	192	85	98,64 %	100,00 %
0,0001	128	300	64	8	64	32	8	6	192	63	98,61 %	100,00 %
0,0001	128	300	64	8	64	32	8	6	192	99	98,60 %	100,00 %
0,0001	128	300	72	8	81	32	8	6	192	120	98,59 %	99,99 %
0,0001	128	300	64	8	64	32	8	12	192	73	98,57 %	99,98 %
0,0001	128	300	72	8	81	32	8	12	192	80	98,56 %	99,99 %
0,0001	128	300	64	8	64	32	8	12	192	103	98,53 %	100,00 %
0,0001	128	300	32	8	16	72	8	6	192	94	98,32 %	100,00 %

Tabla 5.6: Configuración de hiperparámetros con los mejores resultados de ViT con MNIST

5.2.3. Dataset CIFAR-100

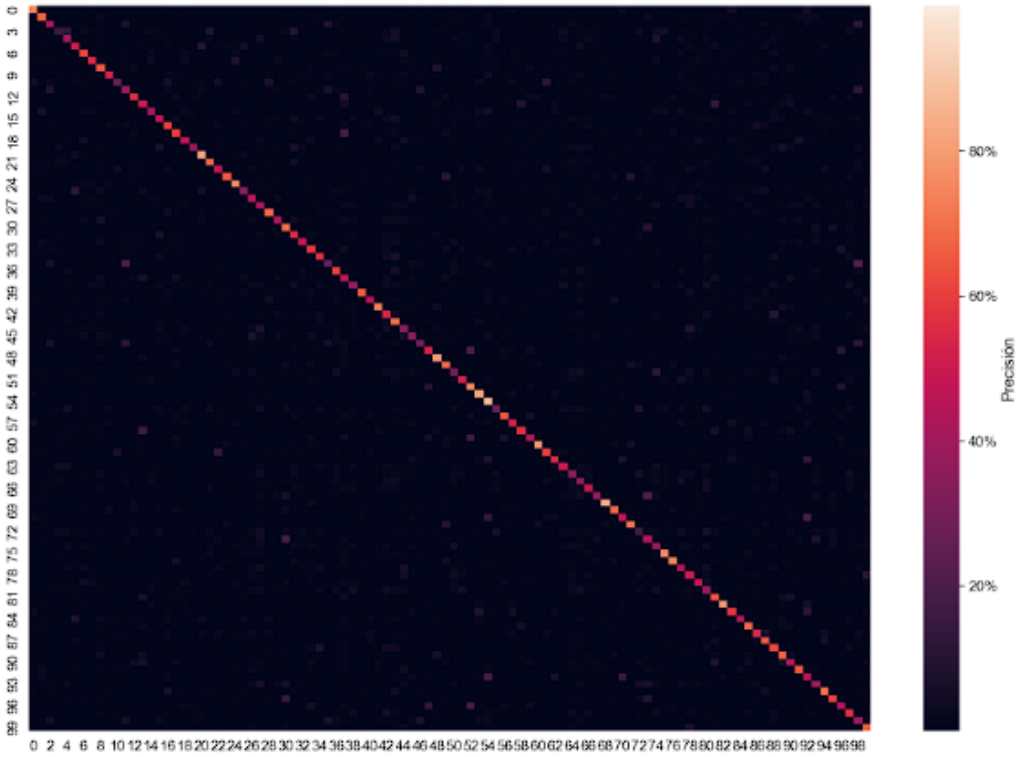

Figura 5.7: Matriz de confusión para CIFAR-100 con ViT

Igual que para gMLP, aparentemente los resultados son buenos, pero al revisar bien el modelo confunde algunas clases, generalmente una clase no está compuesta por las características de muchas otras, sino de una o menos, por lo que es posible que al haber tantas clases y con características similares, el modelo no sepa precisar mucho más.

Al tratarse de un dataset con muchas clases (Tabla 5.7), la cantidad de épocas afecta, requiriendo más de 150 para los mejores resultados, y en su mayoría superan las 200. No por esto más codificadores son mejores, se puede ver que con 40 la precisión es ligeramente menor que con 8 codificadores, y lo mismo ocurre con la cantidad de cabezas que tienen los bloques de MultiHead-Attention, más no significa mejor.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	256	500	72	8	81	64	4	8	192	500	57,97 %	83,47 %
0,0001	256	400	72	8	81	64	4	8	192	190	57,42 %	82,87 %
0,0001	128	400	72	8	81	64	4	8	192	281	56,79 %	83,25 %
0,0001	256	200	72	8	81	64	4	8	192	200	56,77 %	82,94 %
0,0001	256	400	72	8	81	64	4	15	192	147	56,31 %	82,58 %
0,0001	256	400	64	8	64	64	4	8	192	164	56,08 %	82,24 %
0,0001	128	400	72	8	81	72	8	8	192	245	55,94 %	82,02 %
0,0001	256	500	72	8	81	64	4	25	192	500	55,89 %	81,73 %
0,0001	128	400	64	8	64	72	10	8	192	282	55,83 %	82,31 %
0,0001	128	400	64	8	64	128	15	8	192	400	55,76 %	81,43 %
0,0001	128	400	64	8	64	72	8	8	192	299	55,73 %	82,64 %
0,0001	256	400	72	8	81	64	4	40	192	400	55,71 %	81,23 %
5,00E-05	256	400	72	8	81	64	4	8	192	158	55,60 %	82,31 %
0,0001	256	400	72	8	81	64	4	8	192	177	55,45 %	83,41 %
0,0001	128	400	64	8	64	72	15	8	192	263	55,44 %	82,09 %

Tabla 5.7: Configuración de hiperparámetros con los mejores resultados de ViT con CIFAR-100

5.2.4. Dataset FER-2013

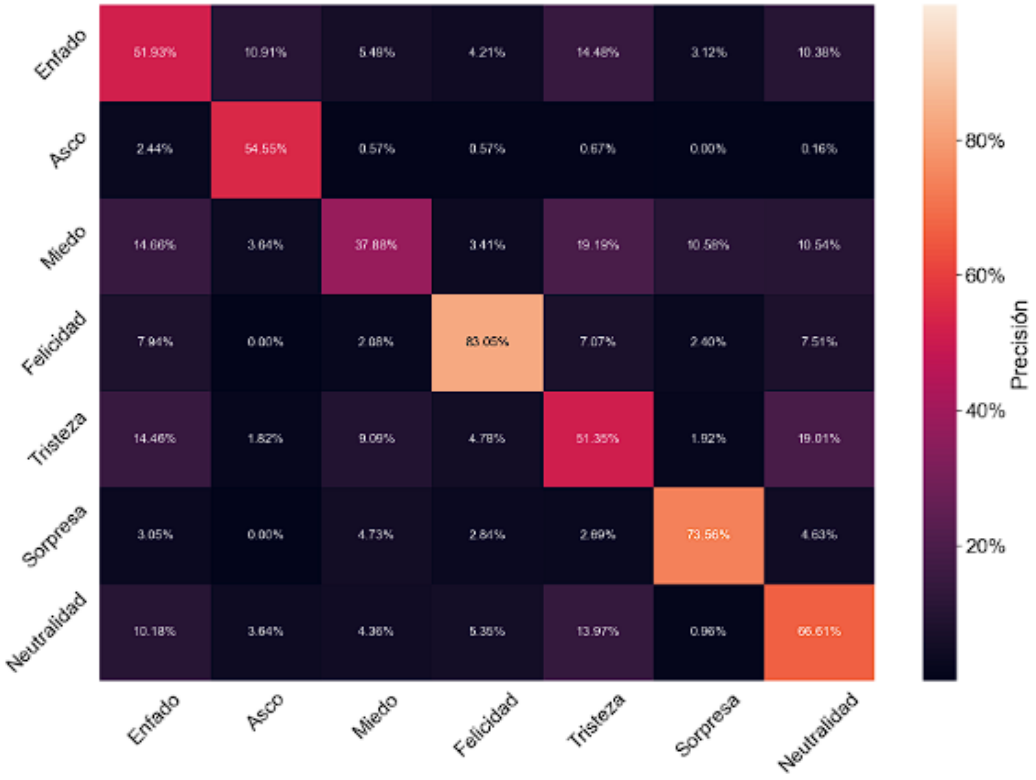

Figura 5.8: Matriz de confusión para FER-2013 con ViT

En comparación a la matriz de gMLP, la de ViT es capaz de clasificar mejor los sentimientos, esparciendo menos las predicciones del enfado y detectando mejor el sentimiento de Asco. Al igual que con el otro modelo, para ViT es difícil clasificar el Miedo en comparación al resto de sentimientos, aunque a pesar de eso da mejores resultados que con gMLP.

Se puede observar (Tabla5.8) que a mayor escalado de la imagen y mayor cantidad de parches por la misma, mejores resultados. Los resultados no mejoran si se utilizan muchas épocas, lo idean es rondar las 100, y de hecho se puede ver como el mejor resultado se estableció con 300 épocas a ejecutar, pero con los mecanismos de prevención de sobreentrenamiento se paró en menos de la mitad de épocas previstas.

Weight decay	Batch size	Epochs	Dropout rate	Image size	Patches size	Patches per image	Embedding dimensions	Number of blocks	Elements per patch	Total epochs executed	Test accuracy	Test top-5 accuracy
0,0001	256	300	72	8	81	64	6	8	192	112	62,50 %	98,58 %
0,0001	128	200	72	8	81	72	8	6	192	112	62,41 %	98,52 %
0,0001	128	200	64	8	64	72	8	12	192	98	61,74 %	98,47 %
0,0001	128	200	64	8	64	32	8	12	192	163	61,72 %	98,47 %
0,0001	128	200	72	8	81	72	8	12	192	90	61,58 %	98,50 %
0,0001	128	200	72	8	81	32	8	6	192	145	61,10 %	98,47 %
0,0001	128	200	72	8	81	72	8	12	192	93	60,96 %	98,33 %
0,0001	128	200	64	8	64	32	8	6	192	160	60,94 %	98,66 %
0,0001	128	200	72	8	81	32	8	12	192	93	60,69 %	98,61 %
0,0001	256	200	64	8	64	32	8	6	192	108	60,38 %	98,61 %
0,0001	128	200	64	8	64	72	8	6	192	94	60,10 %	98,58 %
0,0001	128	200	32	8	16	72	8	12	192	200	55,39 %	97,46 %
0,0001	128	200	32	8	16	72	8	6	192	200	54,89 %	97,60 %
0,0001	256	200	32	8	16	32	8	6	192	167	53,61 %	97,35 %
0,0001	128	200	32	8	16	32	8	12	192	163	52,77 %	97,27 %

Tabla 5.8: Configuración de hiperparámetros con los mejores resultados de ViT con FER-2013

5.3. Comparación de resultados

	ViT				gMLP			
Dataset	***Parámetos***	***Pérdidas***	***Precisión***	***Top 5 Precisión***	***Parámetos***	***Pérdidas***	***Precisión***	***Top 5 Precisión***
CIFAR-10	3.393.130	0,5536	82,37	99,00	1.672.465	0,7460	76,11	98,38
MNIST	3.671.658	0,0554	98,18	99,96	1.271.818	0,0283	99,08	100
CIFAR-100	13.768.036	1,8640	54	80,54	471.268	1,8802	53,36	80,73
FER-2013	13.664.519	1,0977	62	98,58	828.679	1,2899	50,82	97,58

Tabla 5.9: Resultados con datos finales de los dos modelos

Como se puede ver en la tabla superior, la cantidad de parámetros conseguidos frente a las precisiones dadas, marcan una gran diferencia entre el gasto de recursos para obtener salidas similares. Teniendo que los dos modelos generan menos parámetros para los datasets más sencillos, con pocas clases y a color (CIFAR-10) o con formas sencillas (MNIST) hay una diferencia bastante grande de parámetros, duplicando ViT en cantidad y empeorando en los resultados por poco, tanto en la precisión general como en el Top 5 de mejores precisiones.

Ante los datasets más complejos que son CIFAR-100 por su cantidad de clases y FER-2013 por tener que tratar algo tan abstracto como los sentimientos, que además están en blanco y negro; Los dos modelos disminuyen en su rendimiento, aumentando enormemente en ViT los parámetros, incluso más del doble pero situándose mejor en cuanto a resultados para clasificar conjuntos de datos más complejos.

Por último, respecto a la tabla se puede ver la media de pérdidas de cada modelo en el que ocurre algo similar a lo comentado anteriormente. Donde las pérdidas son menores es en MNIST en ambos modelos por la sencillez del dataset, a diferencia de que para CIFAR-100 hay un valor de pérdidas bastante alto por su número de clases. Aunque dentro de los altos valores el más bajo es para FER-2013 con ViT, lo que indica igual que antes que ViT se desenvuelve mejor para la clasificación características de clases más abstractas.

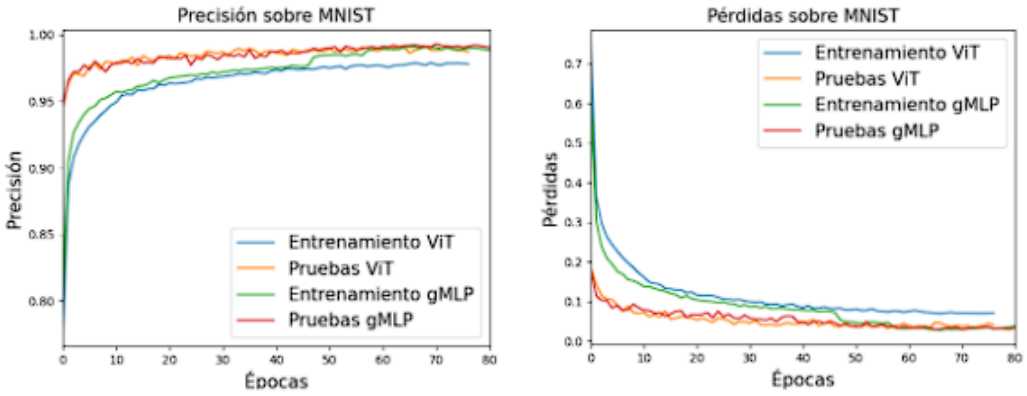

Figura 5.9: Precisión y pérdidas del dataset MNIST

Se puede observar como para MNIST la evolución de precisión de gMLP durante el entrenamiento es mayor, pero cuando se hacen las pruebas ViT en comparación al entrenamiento, dista bastante menos que gMLP, aunque sigue quedando por debajo en cuanto a precisión por época.

En cuanto a las pérdidas, siguen siendo bastante menores en el entrenamiento de gMLP y en las pruebas son casi iguales. Cabe mencionar que para ambos parámetros el entrenamiento y las pruebas de gMLP finalizan siendo bastante similares, mientras que para ViT los resultados son menores siempre en las pruebas que el entrenamiento, dando a entender que este es capaz de alcanzar muy buenos resultados finales aun con los del entrenamiento algo inferiores.

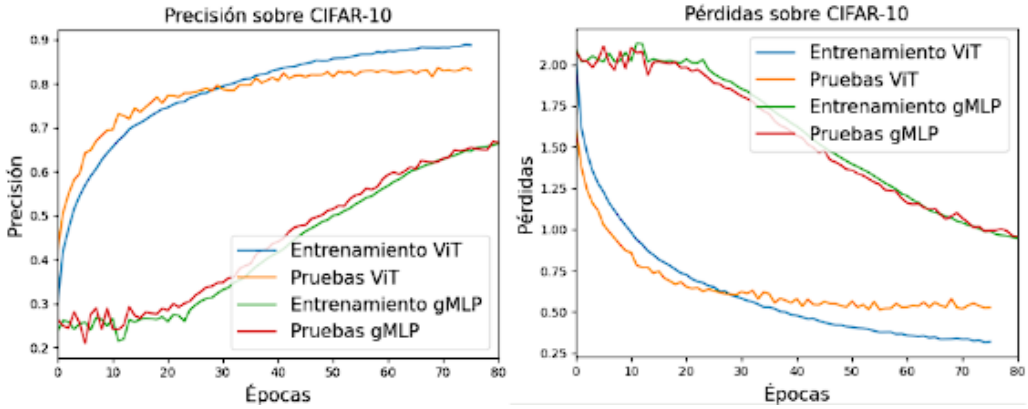

Figura 5.10: Precisión y pérdidas del dataset CIFAR-10

A diferencia de la comparación anterior, que había resultados ciertamente similares, aquí se puede ver claramente cuánto difiere gMLP respecto a ViT. Mientras que desde el comienzo la precisión de ViT mejora considerablemente, estabilizándose cerca de las 30 épocas, a gMLP le cuesta mucho más conseguir precisión en el modelo, consiguiendo relativas mejoras por la época 30 pero sin acercarse en ningún momento a los valores de ViT.

Con las pérdidas pasa lo mismo, a ViT le es mucho más sencillo obtener mejores resultados en épocas cortas, aunque dentro de la comparativa consigo misma de entrenamiento respecto a pruebas, el conjunto de imágenes de prueba da resultados peores que con el entrenamiento. Aunque por supuesto teniendo siempre menos pérdidas que con gMLP y, por tanto, obteniendo mejores resultados.

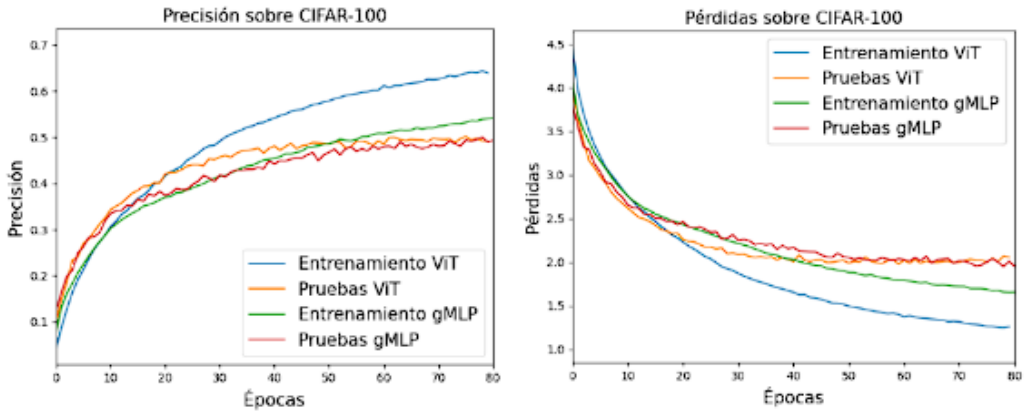

Figura 5.11: Precisión y pérdidas del dataset CIFAR-100

Para el dataset de CIFAR-100 se debe recordar que los resultados son peores que los dos casos anteriores, por ello los ejes no alcanzan valores tan altos en la precisión, ni tan bajos en las pérdidas. Teniendo esto en cuenta, aún sigue en cabeza ViT, con una precisión de entrenamiento notable, mientras que para el conjunto de imágenes de pruebas son resultados más cercanos, como ocurría con MNIST.

Para las pérdidas, lo mismo que la precisión, el entrenamiento de ViT es mejor, alcanza perdidas menores, que en gMLP, pero a la hora ver el registro de pérdidas con los datos de prueba son resultados bastante similares, superando ViT a gMLP por poco.

Con este dataset, por tanto, se concluye que en el entrenamiento ViT supera a gMLP, pero en las pruebas están casi a la par.

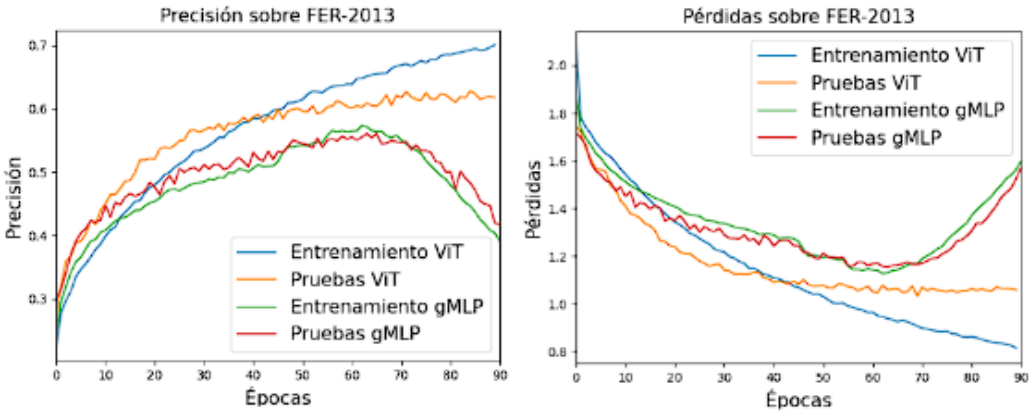

Figura 5.12: Precisión y pérdidas del dataset FER-2013

Para el dataset de conceptos más abstractos hay una clara diferencia de qué modelo se adapta mejor a lo largo de las épocas. Ambos tienen un punto de partida bastante similar, aunque sobre las 20 épocas para el entrenamiento de ViT crece mucho más rápido la precisión y mejoran las pérdidas. En ViT se hace notar el progreso de este cuando alcanzan las 60 épocas donde gMLP empeora drásticamente la precisión, mientras que ViT sigue subiendo, hasta cierto límite claro que es cuando comienza a bajar, pero esto no se muestra gracias a que el modelo para de entrenar cuando prevé el sobreentrenamiento y, por lo tanto, peores resultados.

En cuanto a las pruebas, los resultados son parecidos al entrenamiento, a excepción de que la diferencia es notable a las pocas épocas y la estabilidad de la precisión y pérdidas tanto de ViT como de gMLP comienzan sobre las 30 épocas, habiendo a partir de las 70 una diferencia mayor a la que ya había.

Para finalizar, comparando los resultados de ambos modelos para la clasificación de sentimientos, es destacable que ViT es, no solo bastante mejor respecto a gMLP, sino que ante el deterioro de resultados por sobreentrenamiento, gMLP decae mucho más con cada época.

5.4. Mejores precisiones registradas

Para ultimar las comparaciones, se van a mostrar los mejores resultados de implementaciones propias de otros desarrolladores a día de la realización de este proyecto[13].

Modelo	Precisión	Año de ejecución
ViT-H/14	99,5	2020
ViT-L/16	99,42	2020
CaiT-M-36 U 224	99,4	2021
CvT-W24	99.39	2021
BiT-L	99,37	2019

Tabla 5.10: Resultados de otros modelos para CIFAR-10

Se observa que los resultados son casi perfectos dado que este dataset es de los más utilizados. Los dos primeros puestos, estos lo ocupan unas modificaciones basadas en la arquitectura de Vision Transformer y son resultados dados previos a la publicación de la propia arquitectura. El tercer y cuarto puesto son de la arquitectura Transformer, es decir, usan atención pero no la misma arquitectura que ViT. Y por último BiT-L, basado en ResNet.

Modelo	Precisión	Año de ejecución
Heterogeneous ensemble with simple CNN	99,91	2020
Branching/Merging CNN + Homogeneous Vector Capsules	99.87	2020
EnsNet	99,84	2020
Efficient-CapsNet	99,84	2021
SOPCNN	99,83	2020

Tabla 5.11: Resultados de otros modelos para MNIST

En los resultados de este dataset predominan las redes convolucionales. Abarcan casi todos en mayor o menor medida. Por ejemplo, EnsNet[14], que es una red con partes de CNN y con el objetivo principal de eliminar el texto de las imágenes realizando realiza muy bien la clasificación, o también Efficient-CapsNet, basada la adición de estructuras “cápsula” a las redes convolucionales.

Modelo	Precisión	Año de ejecución
EffNet-L2	96,08	2020
Swin-L + ML-Decoder	95,1	—
ViT-B-16	94,2	—
CvT-W24	94,09	—
ViT-B/16	93,95	—

Tabla 5.12: Resultados de otros modelos para CIFAR-100

Para dar estos resultados, los cinco modelos hicieron uso de datasets adicionales con los que reforzar los entrenamientos. En este caso también vemos que la arquitectura ViT ocupa algunos de los primeros puestos con modificaciones, eso sí.

Modelo	Precisión	Año de ejecución
Ensemble ResMaskingNet with 6 other CNNs	76,82	2013
Local Learning Deep+BOW	75,42	2018
LHC-Net	74,42	2021
Residual Masking Network	74,14	2013
ResNet18 With Tricks	73,70	2021

Tabla 5.13: Resultados de otros modelos para CIFAR-100

Para el último dataset , los modelos utilizados son específicos para estos datos. Es interesante ver que el mejor fue en 2013, aun habiéndose ejecutado nuevas arquitecturas. Con estos datos, se puede ver que aunque sea con modificaciones, la arquitectura Vision Transformer da muy buenos resultados, por lo que sería bastante interesante estudiar en qué mejoran estas modificaciones el modelo base

Capítulo 6

Conclusiones y lineas futuras

El principal objetivo de este trabajo consistió en discernir cuál es el mejor modelo entre Vision Transformer y Gated Multilayer Perceptron. Para ello, ambos modelos fueron implementados bajo las mismas herramientas y fueron probados con los mismos datasets.

Con los datos obtenidos y las comparaciones hechas se puede revelar que Vision Transformer da mejores resultados en todos los casos vistos, teniendo en mente que hace uso de muchos más parámetros que gMLP. Asimismo, y aunque los resultados no distan en gran medida entre sí. Se puede ver como Vision Transformer tiene una progresión más estable y es capaz de comprender conceptos más abstractos relativos a las clases, como son los sentimientos que refleja una persona.

Como líneas futuras para continuar con el desarrollo del trabajo, podrían mejorarse las implementaciones de los modelos, añadiendo mecanismos de aumento automático de las imágenes, donde se seleccionen los mejores tamaños para cada dataset. Además, es importante mencionar que cada modelo tiene distintas variantes, en este proyecto se hizo el estudio con los modelos bases, por lo que sería interesante extender este trabajo implementando y comparando los resultados de las diversas variantes de cada modelo.

Capítulo 7

Summary and Conclusions

The main objective of this work was to discern which is the best model between Vision Transformer and Gated Multilayer Perceptron. For this, both models were implemented under the same tools and were tested with the same datasets.

With the data obtained, and the comparisons made, it can be revealed that Vision Transformer gives better results in all the cases seen, bearing in mind that it makes use of many more parameters than gMLP. Likewise, and although the results are not far from each other. It can be seen how Vision Transformer has a more stable progression and is capable of understanding more abstract concepts related to classes, such as the feelings that a person reflects.

As future lines to continue with the development of the work, the implementations of the models could be improved by adding auto augmentation mechanisms for the images, where the best sizes are selected for each dataset. It is also important to mention that each model has different variants, in this project the study was done with the base models, so it would be interesting to extend this work by implementing and comparing the results of the various variants of each mode

Capítulo 8

Presupuesto

Este capítulo tiene el objetivo de presentar un presupuesto estimado para la ejecución e investigación de los modelos, teniendo en cuenta el equipo utilizado y el coste del investigador/desarrollador que lo haga.

Basado en el salario medio de un Data Scientist con cierta experiencia pero sin llegar a ser Senior, que ronda los 33.000 € anuales, y dando por hecho que la jornada es completa, se estima que el coste de trabajo por hora es 17,19 €.

8.1. Recursos humanos

Queda por tanto la siguiente tabla que refleja los gastos de recursos humanos para la realización de este trabajo, ocupando desde la investigación hasta la implementación y obtención de conclusiones del mismo.

Tarea	**Horas**	**Costes**
Estudio de técnicas Deep Learning	100	1.719 €
Obtención y generación de Datasets	40	687,6 €
Implementación de los modelos y programas adicionales	120	2.062,8 €
Evaluación de resultados	40	687,6 €
TOTAL	**300**	**5.157 €**

Tabla 8.1: Presupuesto de recursos humanos

8.2. Materiales

En cuanto al equipo, se utiliza una máquina alojada en la Universidad de La Laguna a cargo del área de Ingeniería de Sistemas y Automática con dos tarjetas gráficas necesarias para la ejecución del proyecto, alcanzando un coste total de 4.500€.

Más allá de este coste no hubo algún gasto adicional por el uso de herramientas o programas de pago.

Componente	**Costes**
Máquina de la ULL	1.900 €
GPU NVIDIA RTX 2080 Ti	1.000 €
GPU NVIDIA RTX 3090	1.600 €
TOTAL	**4.500 €**

Tabla 8.2: Presupuesto de materiales

8.3. Coste total del proyecto

Queda, por tanto, como coste total de la ejecución del proyecto, a 300 horas de trabajo con un puesto medio de Data Scientist y con el equipo mencionado antes:

Concepto	**Costes**
RRHH	5.157 €
Material	4.500 €
TOTAL	**9.657 €**

Tabla 8.3: Presupuesto total del proyecto

Bibliografía

[1] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. neural computation, 1989.

[2] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge (ilsvrc). Disponible en https://www.image-net.org/challenges/LSVRC/.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Ñ. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. Disponible en https://arxiv.org/abs/1706.03762.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. Disponible en https://arxiv.org/abs/2010.11929.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. Disponible en https://arxiv.org/abs/1607.06450.

[6] Hanxiao Liu, Zihang Dai, David R. So, and Quoc V. Le. Pay attention to mlps, 2021. Disponible en https://arxiv.org/abs/2105.08050.

[7] Joel Veness, Tor Lattimore, David Budden, Avishkar Bhoopchand, Christopher Mattern, Agnieszka Grabska-Barwinska, Eren Sezener, Jianan Wang, Peter Toth, Simon Schmitt, and Marcus Hutter. Gated linear networks, 2019. Disponible en https://arxiv.org/abs/1910.01526.

[8] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. Disponible en http://yann.lecun.com/exdb/mnist/.

[9] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 and cifar-100 datasets. Disponible en https://www.cs.toronto.edu/~kriz/cifar.html.

[10] Fer-2013 from kaggle. Disponible en https://www.kaggle.com/datasets/msambare/fer2013.

[11] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization (adamw), 2017. Disponible en https://arxiv.org/abs/1711.05101.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. Disponible en https://arxiv.org/abs/1412.6980.

[13] Papers with code (models benchmark). Disponible en https://paperswithcode.com/sota.

[14] Shuaitao Zhang, Yuliang Liu, Lianwen Jin, Yaoxiong Huang, and Songxuan Lai. Ensnet: Ensconce text in the wild, 2018. Disponible en https://arxiv.org/abs/1812.00723.

[15] A. Rajagopal and V.Ñirmala. Convolutional gated mlp: Combining convolutions gmlp, 2021. Disponible en https://arxiv.org/abs/2111.03940.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. Disponible en https://arxiv.org/abs/1511.08458.

[17] François Chollet et al. Keras. Disponible en https://keras.io.

[18] Google Brain Team. Tensorflow. Disponible en https://www.tensorflow.org.