SITUACIÓN PROBLEMA: Análisis de Imágenes de Artículos de Vestir con Aprendizaje Profundo

TC2035: Diseño de redes neuronales y aprendizaje profundo

Gpo. 301
Viernes 1 de Diciembre de 2023
Profesor. Dr. Santiago Enrique Conant Pablos
Tecnológico de Monterrey, Campus Monterrey

Valeria María Serna Salazar - **A01284960**

Tecnológico de Monterrey

Índice

I Introducción

En la última década, las redes neuronales y la inteligencia artificial han emergido como tecnologías que han revolucionado diversos campos, desde la automatización industrial y la conducción autónoma de vehículos, hasta el diagnóstico médico asistido por computadora. En particular, las redes neuronales profundas, por su capacidad para simular la estructura del cerebro humano, han demostrado ser instrumentos valiosos para abordar problemas complejos y realizar tareas que antes parecían inalcanzables.

Aunque se propusieron algunos modelos inspirados en la arquitectura del sistema nervioso en 1950, no fue sino hasta las últimas dos décadas que el resurgimiento de las redes neuronales profundas, impulsado por avances en hardware y grandes conjuntos de datos, permitió desbloquear su verdadero potencial. La implementación de arquitecturas como las Redes Neuronales Convolucionales (CNN) y las Redes Neuronales Recurrentes (RNN) marcó un antes y un después en la historia de la computación, llevando a mejoras sustanciales en dicho campo.

Con esto en mente, el problema de la clasificación de imágenes ha sido un terreno fértil para la aplicación de las redes neuronales. La capacidad de aprender patrones complejos y representar información de manera jerárquica ha permitido avances notables en la identificación y clasificación de objetos en imágenes.

La base de datos a utilizar durante la realización de este proyecto es Fashion MNIST, una versión alternativa al clásico conjunto de datos MNIST de dígitos escritos a mano. Esta cuenta con 70,000 imágenes de 28x28 píxeles, distribuidas en 10 clases que representan diversas prendas de vestir, como camisetas, pantalones, vestidos, entre otros. El objetivo es poner a prueba la capacidad de los modelos de aprendizaje automático en tareas más complejas y realistas, que signifiquen un aprendizaje aplicado a la vida diaria.

En este contexto, la implementación de redes neuronales específicas se vuelve crucial. El uso de arquitecturas especializadas, como LeNet-5 y redes neuronales siamesas, permite adaptar los modelos a las características únicas del conjunto de datos en cuestión. Cabe aclarar que la elección de estas arquitecturas se basa en su capacidad para capturar patrones espaciales en imágenes y aprender representaciones eficientes, respectivamente. Al final, no solo se busca evaluar la precisión de los modelos, sino también comprender cómo estas redes especializadas pueden ofrecer soluciones más efectivas en la clasificación de prendas de vestir.

II LeNet-5 (Visto en Clase)

LeNet-5 es una red neuronal convolucional (CNN) desarrollada por Yann LeCun, Léon Bottou, Yoshua Bengio y Patrick Haffner en 1998. Su objetivo principal era reconocer dígitos escritos a

mano y fue utilizada principalmente en la tarea de reconocimiento de caracteres en cheques bancarios. Sin embargo, su arquitectura influyó en el desarrollo de redes más avanzadas para tareas de visión por computadora. Esta red fue una de las primeras en utilizar capas convolucionales, que son especialmente efectivas para procesar datos estructurados, como imágenes. Además, la red usa capas de submuestreo (pooling) para extraer características principales y reducir la dimensionalidad de los datos, así como capas completamente conectadas para realizar la clasificación final. Adicionalmente, utiliza funciones de activación no lineales, como la tanh y la sigmoide, aparte de usar la función softmax en la capa de salida para producir una distribución de probabilidad sobre varias clases.

Una de las razones principales para elegir este modelo como el primero a utilizar es que tiene una arquitectura relativamente simple, lo que puede ser beneficioso para conjuntos de datos menos complejos, como Fashion MNIST, que solo requiere catalogar 10 clases. Además, es más ligera en términos de parámetros y operaciones, lo que indica un menor gasto de recursos computacionales. Finalmente, este modelo ha demostrado ser efectivo en la extracción de características, lo que podría ser útil para reconocer patrones relevantes en imágenes de moda.

La manera en que funciona el modelo es con 7 capas. En la figura 1 se muestra el diagrama de su funcionamiento, pero en detalle, funciona de la siguiente manera:

Capa de entrada (C1):

- Recibe la imagen original; cada filtro convolucional en esta capa se encarga de buscar patrones locales en la imagen, como bordes, esquinas o texturas.
- Los filtros aplican operaciones de convolución, multiplicando sus pesos con los píxeles correspondientes de la imagen y sumando los resultados para producir mapas de características convolucionales.

Capa de submuestreo (S2):

- Después de la convolución, la capa de submuestreo reduce la dimensionalidad del mapa de características resultante.
- Esto se realiza mediante operaciones de submuestreo (como el max-pooling), que seleccionan el valor máximo de un conjunto de píxeles vecinos en el mapa de características. De esta manera, se preservan las características más importantes y se reduce la cantidad de datos a procesar.

Capa convolucional (C3):

- Similar a la primera capa convolucional, la capa C3 aplica nuevos filtros para detectar patrones más complejos y abstractos en las características extraídas por la capa S2.

- La aplicación de funciones de activación no lineales, como la tangente hiperbólica, introduce no linealidades y permite a la red aprender representaciones más complejas.

Capa de submuestreo (S4):

- Al igual que S2, la capa S4 realiza submuestreo para reducir la dimensionalidad del mapa de características resultante de la capa C3.
- Esto ayuda a centrarse en las características más relevantes y reduce la cantidad de parámetros en las capas posteriores.

Capa totalmente conectada (C5):

- Toma las características de la capa S4 y las conecta completamente, transformándolas en un vector de características para la clasificación.
- Utiliza funciones de activación no lineales para introducir no linealidades y aprender representaciones no lineales de las características.

Capa de salida (F6):

- Utiliza una función de activación softmax para convertir las salidas en probabilidades de pertenencia a cada clase.
- Durante el entrenamiento, se compara la salida de la red con las etiquetas reales, y se ajustan los pesos de la red mediante el algoritmo de retropropagación para minimizar la función de pérdida.

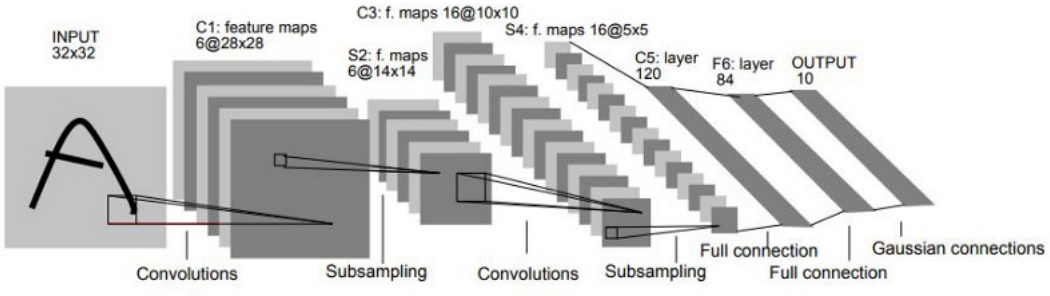

Figura 1: Diagrama de función de LeNet-5

Para iniciar con el código, es necesario mencionar que se tomó como base el visto en clase, que utilizaba originalmente el conjunto de datos MNIST. Entonces, el primer paso fue importar las librerías necesarias, así como cargar la base de datos de Fashion MNIST directamente de los datasets de Keras. Después, se imprimieron las dimensiones de los arreglos y se obtuvo que x_train

tiene 60,000 imágenes de 28x28 pixeles, x_test tiene 10,000 imágenes de 28x28 pixeles, y_train tiene 60,000 etiquetas y y_test tiene 10,000 etiquetas.

Después, se imprimió una imagen correspondiente a cada clase para revisar que todo funcione de manera correcta. El resultado está en la figura 2.

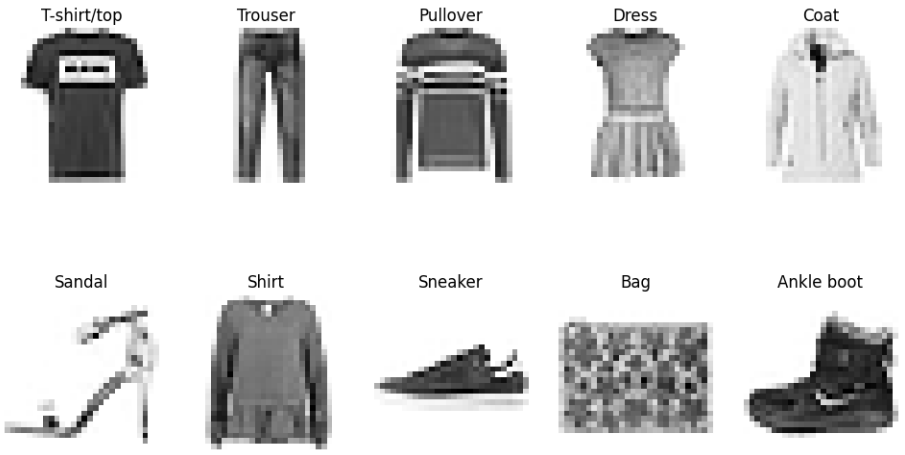

Figura 2: Un artículo correspondiente a cada clase

Adicionalmente, se normalizaron las intensidades de los píxeles para tener valores entre 0 y 1. Así, fue posible hacer una conversión de etiquetas de clase a código binario con one-hot encoding, para ayudar al modelo a comprender mejor la relación entre clases.

Por último, antes de crear el modelo se hizo un reformateo de las imágenes, agregando una dimensión para el canal de escala de grises y así lograr que sea compatible.

La red creada consta de las siguientes capas:

- **C1 (Capa convolucional):** Realiza un filtrado con 32 kernels (filtros) de tamaño 3x3, además de utilizar una función de activación ReLU.
- **S2 (Capa de submuestreo):** Realiza una capa de Max Pooling con un pool size de 2x2, tiene pasos de 1 en ambas dimensiones y tiene un padding de 'valid', lo que significa que no añade relleno a la entrada antes de hacer el pooling.
- **C3 (Capa convolucional):** Realiza un filtrado con 64 kernels (filtros) de tamaño 3x3, además de usar una función de activación ReLU.
- **S4: Capa de submuestreo:** Realiza una capa de Max Pooling de 2x2, tiene pasos de 2 en ambas direcciones con un padding 'valid'.
- **Dropout:** Posee una capa de dropout con una tasa de abandono del 60%, se utiliza para prevenir el sobreajuste eliminando aleatoriamente conexiones entre las neuronas.

- **C5 (Flatten & Fully Conection):** Capa de aplanado para convertir la salida de las capas en un vector plano. A continuación, se encuentra una capa completamente conectada con 256 neuronas y función de activación ReLU, finalmente, una capa de normalización por lotes para estabilizar el entrenamiento y una capa de Dropout con tasa de abandono del 30%.
- **F6 (Capa completamente Conectada):** Esta capa tiene 128 neuronas y función de activación ReLU, además de la Batch Normalization y el Dropout con tasa de abandono del 30%.
- **Capa de salida:** Completamente conectada con 10 neuronas (10 clases), utiliza una función de activación Softmax para obtener probabilidades de pertenencia a cada clase.

De hecho, el diagrama del modelo utilizado se encuentra en la figura 3.

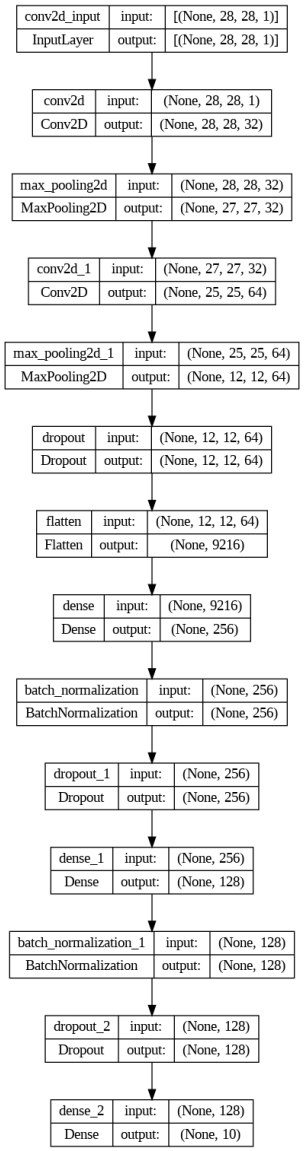

Figura 3: Arquitectura utilizada

Adicionalmente, los hiperparámetros que se definieron y mostraron un comportamiento superior en comparación con otros intentos fueron 75 épocas, 256 batches, y un learning rate de 0.00001. Además, se utilizó un optimizador Adam con parámetros beta_1 y beta_2 de 0.9 y 0.999, respectivamente. Estos valores indican los coeficientes de decaimiento para el promedio móvil de primer y segundo orden.

Asimismo, se presentan las curvas de aprendizaje mediante ggplot (figura 4 y 5), que muestran un aprendizaje bastante sólido de los datos.

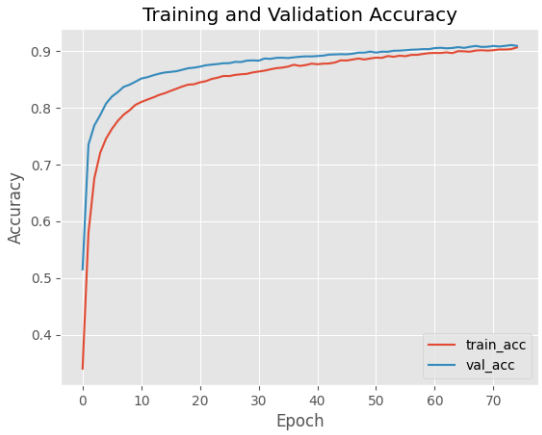

Figura 4: Training Accuracy vs Validation Accuracy

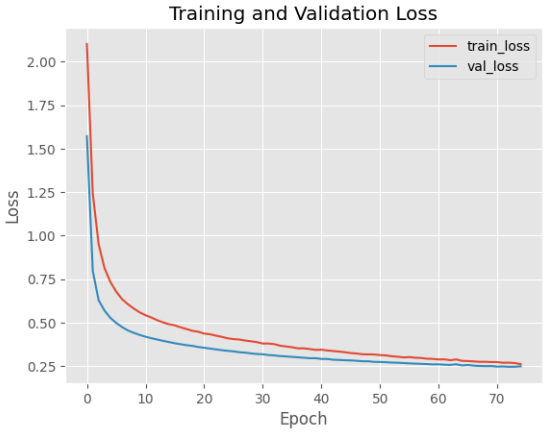

Figura 5: Training loss vs Validation loss

Finalmente, se muestran los valores de desempeño de la red. Estos números fueron considerados para la mejor elección de los hiperparámetros.

En primer lugar, se presenta el accuracy, que es la proporción de predicciones correctas con respecto al total de predicciones. Su fórmula es:

(1) $$\boxed{\frac{(True_{pos} + True_{neg})}{(True_{pos} + False_{pos} + True_{neg} + False_{neg})}}$$

Luego, se muestra la precisión, que es la proporción de predicciones positivas correctas con respecto al total de predicciones positivas. Su objetivo es medir la capacidad del modelo para no etiquetar incorrectamente como positivos. Su fórmula es:

(2) $$\boxed{\frac{True_{pos}}{(True_{pos} + False_{pos}})}$$

El siguiente valor a imprimir es el recall, que es la proporción de instancias positivas que fueron correctamente predichas con respecto al total de instancias positivas. Este se encarga de medir la capacidad del modelo para identificar correctamente todas las instancias positivas. Su fórmula es:

(3) $$\boxed{\frac{True_{pos}}{(True_{pos} + False_{neg}})}$$

El último valor a imprimir es el F1-Score, que es la media entre la precisión y el recall. Es útil cuando hay un desequilibrio entre las clases, y un número alto indica un buen equilibrio entre la precisión y la capacidad para recuperar instancias positivas. Su fórmula es:

(4) $$\boxed{\frac{2 * (Precision * Recall)}{(Precision + Recall)}}$$

El resultado final fue un Accuracy del 91%, con una precisión y recall promedio también del 91%. El desglose de estas métricas por clase proporciona una comprensión completa del modelo. Por ejemplo, en la figura 6 es posible observar que las clases que el modelo predice mejor son Trouser, Sandals, Sneaker, Bag y Ankle Boots, mientras que aquellas con una menor precisión y recall son Pullover, Coat y Shirt, lo cual tiene sentido ya que esas imágenes tienden a tener similitudes entre sí.

Aunque el 91% fue el accuracy más alto que se obtuvo sin overfitting, se llevaron a cabo una serie de experimentos (más de 75) variando la cantidad de capas y los valores de los hiperparámetros. Algunos de los intentos se muestran en la figura 7.

Entonces, se realizaron modificaciones en la arquitectura de la red, los hiperparámetros y las funciones de activación y optimizadores a partir del código base, con el objetivo de obtener el

	precision	recall	f1-score	support
T-shirt/Top	0.86	0.88	0.87	1000
Trouser	0.99	0.97	0.98	1000
Pullover	0.80	0.90	0.85	1000
Dress	0.90	0.92	0.91	1000
Coat	0.88	0.81	0.84	1000
Sandals	0.98	0.98	0.98	1000
Shirt	0.77	0.71	0.74	1000
Sneaker	0.95	0.97	0.96	1000
Bag	0.98	0.98	0.98	1000
Ankle Boots	0.98	0.96	0.97	1000
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Figura 6: Estadísticas de desempeño

Learning Rate	Num_batches	Epochs	Activation	Accuracy
0.01	128	50	tanh	78%
0.001	256	75	tanh	81%
0.001	256	50	ReLU	88%
0.0001	256	75	ReLU	91%
0.0005	128	75	ReLU	87%

Figura 7: Experimentos variando valores

mejor resultado posible. Cabe aclarar que, aunque hubo valores de accuracy más altos, estos surgían de un modelo con overfitting.

En conclusión, durante la ejecución de este código y modelo para abordar el desafío de Fashion MNIST, se logró confirmar la eficacia de las redes convolucionales como una de las mejores opciones para analizar imágenes, especialmente en contextos donde las relaciones locales son cruciales. Con un modelo que tomó 5 minutos para ejecutar 75 épocas, o 15 épocas por segundo, demostró ser eficiente y eficaz. Al obtener un accuracy del 91%, se destaca la idoneidad de este enfoque para problemas complejos de clasificación de imágenes, en este caso, la identificación de prendas de vestir.

Una particularidad del modelo con el dataset utilizado es la confusión entre las clases que se parecen un poco, como vestidos y camisas. Esto podría resolverse de varias maneras, incluyendo la creación de una red más compleja o la búsqueda de métodos para agrupar los datos de manera que las clases no sean tan similares. Sin embargo, este desafío ya se había anticipado desde antes, y a pesar de ello se logró un accuracy de predicción bastante bueno. En última instancia, el modelo funciona según lo esperado, proporcionando una buena predicción para cada artículo de ropa, como se puede observar en la figura 8.

III Red Neuronal Siamesa (Investigado)

La red neuronal siamesa, inspirada en el antiguo concepto de gemelos siameses, es un tipo especializado de arquitectura de redes neuronales que implica la comparación de la similitud entre datos. Surgió inicialmente para abordar problemas de reconocimiento facial, donde se busca determinar si

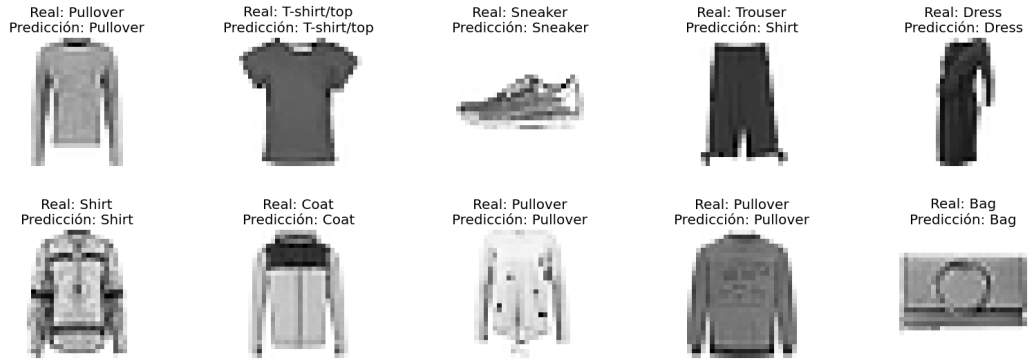

Figura 8: Predicciones usando LeNet-5

dos imágenes representan la misma persona. Sin embargo, su versatilidad ha llevado su aplicación a diversas áreas, como el procesamiento de lenguaje natural, el análisis de similitud en texto y la identificación de patrones en conjuntos de datos complejos.

Se eligió este modelo con la tarea principal de identificar, a partir de una imagen dada, aquellas que presentan la menor distancia (indicando similitud y probable pertenencia a la misma clase). Entonces, aunque no se trata de un modelo destinado a realizar predicciones explícitas, su elección se fundamenta en la capacidad de discernir si puede encontrar imágenes similares a la inicial.

Aunque más adelante se explicará en detalle lo realizado en el código, como se muestra en la figura 9, en general funciona de la siguiente manera:

1. **Entrada gemela:** Ambas ramas reciben datos de entrada simultáneamente.

2. **Capas Compartidas:** Ambas ramas pasan a través de capas compartidas, las cuales tienen la misma arquitectura y comparten los mismos pesos. El propósito es aprender representaciones semánticas comunes para las entradas.

3. **Capas Individuales:** Después de las capas compartidas, cada rama puede tener capas adicionales que son exclusivas de esa rama. Así, cada rama se puede ajustar a las características específicas de su entrada.

4. **Comparación de Salidas:** Las salidas de ambas ramas (las representaciones aprendidas) se comparan para evaluar la similitud entre las entradas. Esto se hace mediante el cálculo de la distancia.

5. **Función de Pérdida:** Durante el entrenamiento se utiliza una función de pérdida para medir la discrepancia entre las salidas y guiar el ajuste de los pesos, de manera que las representaciones aprendidas sean más similares para entradas similares y menos similares para entradas diferentes.

6. **Entrenamiento Conjunto:** Ambas ramas se entrenan simultáneamente con los mismos datos de entrada. Los pesos compartidos se ajustan para minimizar la pérdida global.

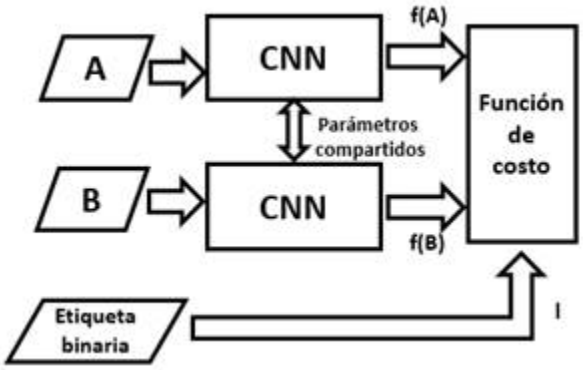

Figura 9: Arquitectura de la Red Neuronal Siamesa

En cuanto al código, se partió de un tutorial creado por Pere Martra, pero se modificó y adaptó de manera significativa para hacerlo funcionar en este contexto principal.

El preprocesamiento realizado sobre los datos es muy similar al realizado con el otro modelo. Se leen los datos desde los datasets de Keras, se obtienen los conjuntos de datos y se dividen entre 255 para normalizarlos. Después, se imprime una imagen de cada clase, obteniendo el mismo resultado que se muestra en la figura 2.

Antes de crear el modelo, se realizan algunas acciones. Primero, se crea una función llamada create_pairs que busca formar pares entre el conjunto de datos. Esta función consiste en seleccionar aleatoriamente otro elemento de la misma clase que el elemento actual del conjunto de datos, asegurando que el par contenga elementos de la misma categoría. Luego, se asignan etiquetas a los pares, indicando si los elementos son iguales (1) o diferentes (0). Si son iguales, se incrementa el contador para verificar si se ha cumplido el número mínimo y, cuando finalmente se cumple, se selecciona aleatoriamente cualquier elemento del conjunto de datos para formar el par, asegurando variedad en la muestra. Finalmente, la función devuelve los pares generados y las etiquetas asociadas.

Después, se establece un límite de 2,000 para la cantidad de datos en el conjunto de validación. Se toman los primeros 2,000 elementos del conjunto de entrenamiento original para formar el conjunto de validación (X_val y y_val). Los elementos restantes se utilizan para crear un conjunto de entrenamiento modificado (X_train2 y y_train2). La idea detrás de esto es separar un conjunto de datos más pequeño para poder validar el rendimiento del modelo antes de la fase de prueba.

Además, se generan los pares de entrenamiento y validación utilizando la función mencionada anteriormente (create_pairs). Para el conjunto de entrenamiento X_train2 y y_train2 se crean

pares con un requisito mínimo de 15,000 pares que contengan elementos de la misma clase. Para X_val y y_val se generan pares con un requisito mínimo de 800 pares iguales.

Como último paso antes de crear el modelo, se definen algunas funciones que serán utilizadas. Primero, la función euclidean_distance que calcula la distancia euclidiana entre dos vectores, x e y, devolviendo como resultado la métrica de distancia euclidiana. También está la función eucl_dist_output_shape que devuelve la forma de salida de la función anterior. Por último, la función contrastive_loss_with_margin que define una función de pérdida contrastiva para entrenar el modelo siamese con un margen específico.

En cuanto al modelo, primero se define una función que crea al modelo base (llamada initialize_base_branch()) que consta de 3 capas densas, cada una seguida por una capa de dropout para evitar el sobreajuste. Entonces, el modelo base tiene:

1. Capa densa con 256 neuronas y activación ReLU.
2. Capa de Dropout con tasa del 30%.
3. Capa densa con 128 neuronas y activación ReLU.
4. Capa de Dropout con una tasa del 30 %.
5. Capa densa con 64 neuronas y activación ReLU.

Después, se utilizan las funciones anteriores para agregar 3 capas adicionales al modelo base, con el objetivo de construir una red siamesa.

1. Agrega una capa de entrada para la parte izquierda del par de entrada, la cual se conecta a la rama base.
2. Agrega una capa de entrada para la parte derecha del par de entrada, la cual se conecta a la misma rama base.
3. Utiliza una capa lambda para calcular la distancia euclidiana entre las representaciones aprendidas de ambas entradas.

La arquitectura final del modelo creado se observa en la figura 10.

Una vez planteada la arquitectura de la red, se definen los hiperparámetros y parámetros necesarios. Es importante mencionar que estos valores se determinaron como la mejor opción para obtener resultados óptimos en este problema específico. Se lograron a través de pruebas y errores al experimentar con diferentes configuraciones y observar los valores de Accuracy, Recall y F1 Score obtenidos.

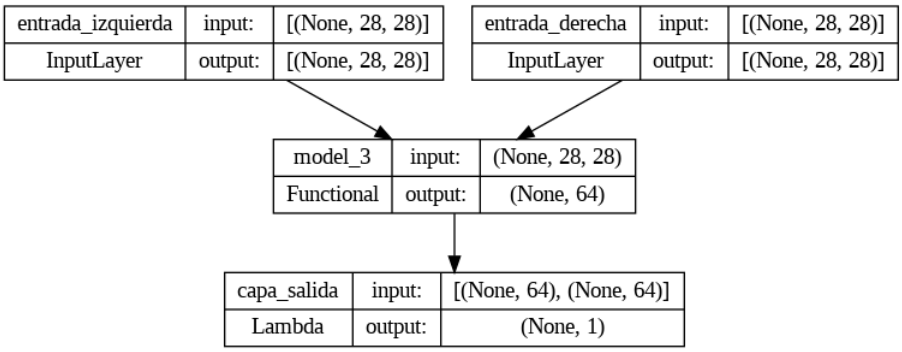

Figura 10: Arquitectura de la Red Neuronal Siamesa creada

Al final, se crea un modelo que utiliza el optimizador RMSprop (que adapta la tasa de aprendizaje para cada parámetro por separado), con la función de pérdida definida anteriormente, durante 20 épocas, con un batch_size de 128 y la opción de validation_data para evaluar el modelo posteriormente.

Para la evaluación, se utiliza una función llamada compute_accuracy() que asume que si la diferencia entre las predicciones y las etiquetas verdaderas es menor a 0.4, entonces son consideradas iguales. Con esta consideración, se calculan las métricas de evaluación y se obtiene un Accuracy del 92%, Validation Accuracy del 89%, Precision del 93%, Recall del 85% y F1 Score del 89%. Estos resultados indican que el modelo está funcionando de manera bastante efectiva.

Finalmente, se imprime una imagen aleatoria y, junto a ella las 3 con una distancia menor (significando que son las más parecidas y, por ende, probablemente pertenecen a la misma clase). Haciendo pruebas con varios objetos se obtienen las figuras 11, 12, 13, 14, 15

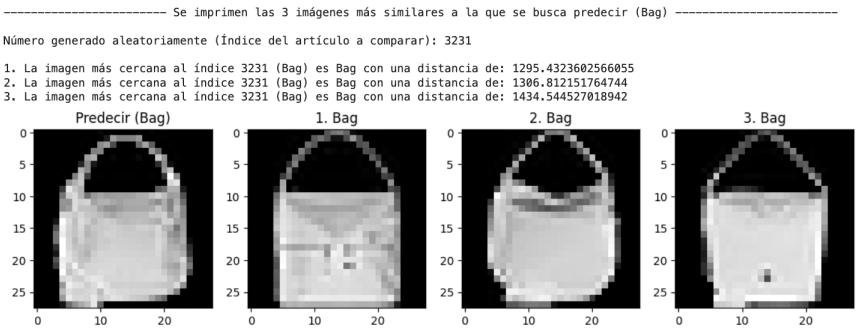

Figura 11: Número aleatorio e imágenes más similares

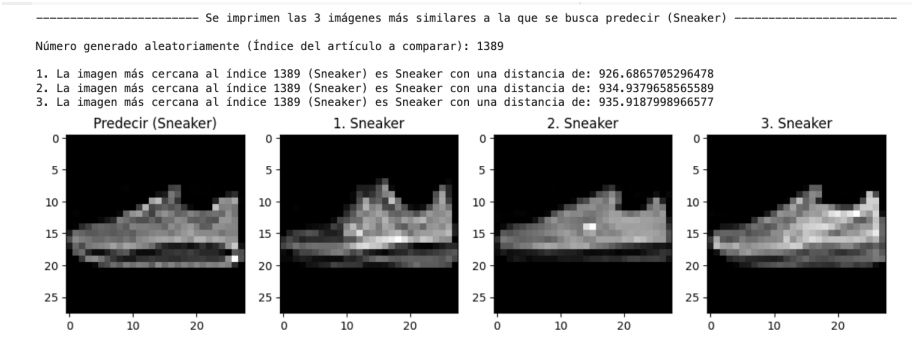

Figura 12: Número aleatorio e imágenes más similares

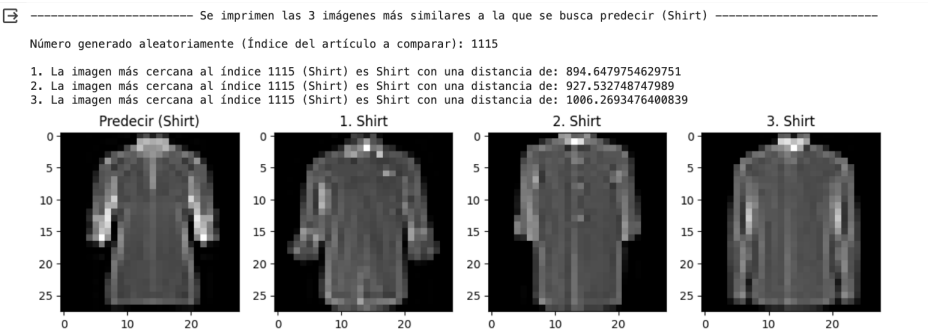

Figura 13: Número aleatorio e imágenes más similares

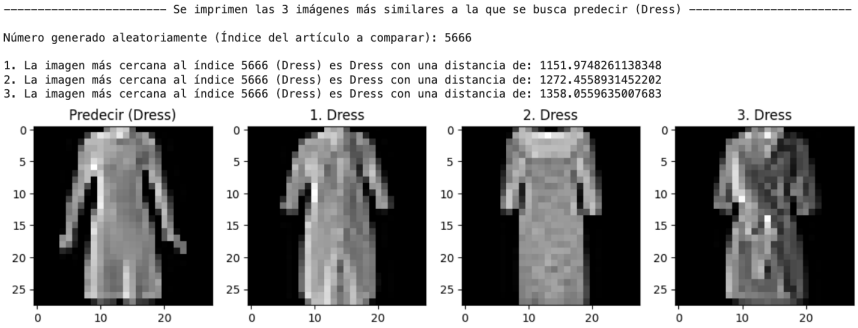

Figura 14: Número aleatorio e imágenes más similares

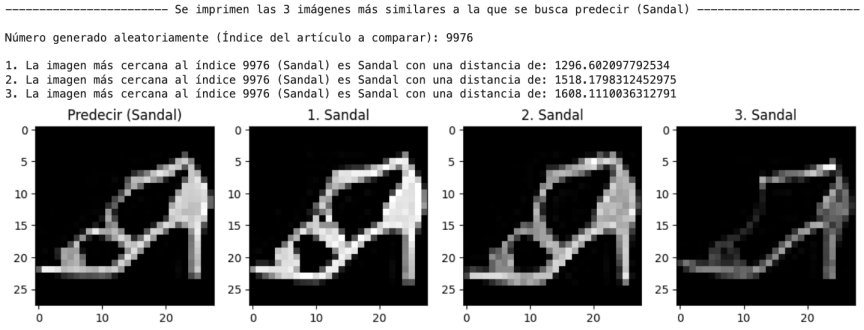

Figura 15: Número aleatorio e imágenes más similares

De hecho, el programa funciona mejor de lo esperado, incluso cuando hay errores en las imágenes con mayor similitud, en realidad sí se parecen entre sí. Por ejemplo, en la figura 16 aparece un "coat" en lugar de un "dress", sin embargo, incluso un humano podría cometer ese error. Entonces, este modelo tiene una capacidad muy buena para encontrar imágenes que le parecen y creo que puede llegar a ser una herramienta poderosa. Sin embargo, creo que las peculiaridades que se presentan son las mismas que se mencionaron con el otro modelo; el tener clases que se parecen tanto, como "t-shirt" y "shirt", podría significar un deterioro del Accuracy. Entonces, aunque el modelo es bastante bueno, se asume que si se dispusiera de más tiempo para encontrar una manera de agrupar las clases o identificar las diferencias entre ellas, el modelo funcionaría de una mejor manera.

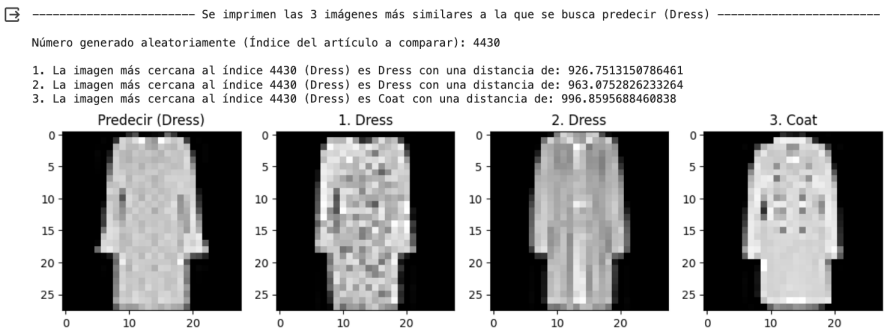

Figura 16: Número aleatorio e imágenes más similares

Cabe aclarar que hubo muchas modificaciones al código original, desde la arquitectura y los hiperparámetros, hasta la adición de la manera de visualizar las predicciones y evaluar el modelo con diversas métricas.

En conclusión, las ventajas de utilizar una red siamesa se hicieron evidentes, ya que el aprendizaje de similitud y la transferencia de conocimiento permitieron que en poco tiempo se construyera una red bastante compleja.

IV Comparación de ambos modelos

En la comparación entre los modelos LeNet-5 y la red neuronal siamesa, considero que el segundo resultó más desafiante de entender debido a su arquitectura compleja y la necesidad de comprender la codificación de los pares y el cálculo de la distancia. Sin embargo, una vez que se estableció un código base y se investigó la teoría detrás de este, la adaptación y optimización se volvieron más sencillas.

LeNet-5, con un learning rate de 0.00001 y 75 épocas, exhibió una ejecución prolongada de casi 6 minutos en la GPU T4 de Google Colaboratory. En contraste, la red siamesa, entrenada con 20

épocas y un learning rate variable, se ejecutó en aproximadamente un minuto, logrando no solo un tiempo significativamente menor sino también una mayor precisión.

Tomando en cuenta el tiempo computacional, las métricas de desempeño y los resultados generales, aunque no hace predicciones explícitas sobre la clase que le corresponde, considero que el mejor modelo fue la red siamesa.

V Conclusiones

En conclusión, la similitud entre las clases 'T-shirt/top' y 'Shirt' plantea un desafío considerable para el modelo, generando confusiones que podrían afectar su desempeño. Explorar opciones para modificar estas clases es esencial para mejorar la precisión y la interpretabilidad del modelo, aunque la solución más factible sería hacer una red más compleja que pueda reconocer estos patrones. Sin embargo, un 92% de Accuracy es bastante positivo, lo que indica que no es algo urgente de solucionar y ambos modelos funcionan de una manera muy buena.

Durante el proyecto, adquirí valiosos conocimientos sobre la implementación de modelos de redes neuronales para la clasificación de imágenes, centrándome específicamente en la resolución del conjunto de datos Fashion MNIST al utilizar LeNet-5 y una red siamesa.

Lo que más me gustó de este proyecto fue la simulación de un entorno profesional en el que pude aplicar técnicas avanzadas de inteligencia artificial. Este ejercicio no solo fortaleció mis habilidades técnicas, sino que también despertó un gran interés en el potencial que la inteligencia artificial y las redes neuronales tienen para transformar el futuro; observar cómo los modelos aprenden y mejoran su rendimiento a medida que se enfrentan a conjuntos de datos complejos es fascinante y abre una expectativa a lo que viene.

Sin embargo, no todo fue un camino sin obstáculos. En particular, me encontré con la mayor parte de los desafíos al hacer la red siamesa, pues la información discernible en la web sobre este enfoque es limitada, por lo que la falta de recursos claros y detallados iniciales complicó la comprensión de la arquitectura y su implementación. A pesar de esto, logré superar el reto mediante una combinación de práctica, prueba y error, y una investigación profunda sobre el tema. De tal manera, el proyecto no solo amplió mi comprensión de las redes neuronales, sino que también me enseñó la importancia de la perseverancia y la capacidad de adaptación en el campo de la inteligencia artificial.

VI Referencias Bibliográficas

[IBM, sf] (s.f). ¿qué son las redes neuronales convolucionales? `https://www.ibm.com/mx-es/topics/convolutional-neural-networks`.

[Bangar, 2022] Bangar, S. (2022). Lenet 5 architecture explained. `https://medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b`.

[Bertinetto et al., 2016] Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. (2016). Fully-convolutional siamese networks for object tracking. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part II 14*, pages 850–865. Springer.

[Boden, 1996] Boden, M. A. (1996). *Artificial intelligence*. Elsevier.

[Daudt et al., 2018] Daudt, R. C., Le Saux, B., and Boulch, A. (2018). Fully convolutional siamese networks for change detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 4063–4067. IEEE.

[Guo et al., 2017] Guo, Q., Feng, W., Zhou, C., Huang, R., Wan, L., and Wang, S. (2017). Learning dynamic siamese network for visual object tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 1763–1771.

[Kayed et al., 2020] Kayed, M., Anter, A., and Mohamed, H. (2020). Classification of garments from fashion mnist dataset using cnn lenet-5 architecture. In *2020 international conference on innovative trends in communication and computer engineering (ITCE)*, pages 238–243. IEEE.

[Wei et al., 2019] Wei, G., Li, G., Zhao, J., and He, A. (2019). Development of a lenet-5 gas identification cnn structure for electronic noses. *Sensors*, 19(1):217.

[Zhang et al., 2019] Zhang, C.-W., Yang, M.-Y., Zeng, H.-J., and Wen, J.-P. (2019). Pedestrian detection based on improved lenet-5 convolutional neural network. *Journal of Algorithms & Computational Technology*, 13:1748302619873601.