Instituto Superior Politécnico Córdoba

Tecnicatura Superior en Innovación con Tecnologías 4.0

Desarrollo de Sistemas de Inteligencia Artificial

Trabajo Práctico Final

Profesor: Sol del Valle Figueroa

Cohorte: 2023

Alumnos: Cristian Ghisiglieri
Naylui Osuna
Romina Peña

Correos: cristian.g2011@gmail.com
nayluiosuna@gmail.com
rominabpena@gmail.com

Tabla de Contenido

Introducción

En el año 2011 derivó de Machine Learning una rama llamada Deep Learning o Aprendizaje Profundo, se dice que es el nueva evolución del machine learning, se trata de un algoritmo automático que imita la percepción humana inspirada en nuestro cerebro y la conexión entre neuronas. El Deep Learning es la técnica que más se acerca a la forma en la que aprendemos los humanos.

En la última década, la inteligencia artificial ha evolucionado significativamente, con el Deep Learning emergiendo como una de sus ramas más prometedoras. Este enfoque ha revolucionado la manera en que las máquinas procesan y comprenden la información visual, superando ampliamente a los métodos tradicionales. El proyecto que presentamos se enfoca en la aplicación práctica de redes neuronales profundas para la clasificación de imágenes, empleando el conjunto de datos Fashion-MNIST como caso de estudio. Fashion-MNIST, es una gran base de datos de imágenes de moda de libre acceso, conocido por su representatividad y equilibrio entre simplicidad y complejidad, ofrece un desafío adecuado para probar y optimizar modelos de aprendizaje profundo. La meta es no sólo explorar el funcionamiento de estas redes, sino también evaluar su eficacia y adaptabilidad en el reconocimiento de patrones visuales, sentando así las bases para futuras aplicaciones en visión computacional.

"Desarrollo e Implementación de un Sistema de Clasificación de Imágenes mediante Deep Learning utilizando Fashion-MNIST durante el período 2023-2024"

El campo de la Inteligencia Artificial ha experimentado avances significativos en los últimos años, particularmente en el área de Deep Learning y su aplicación en la clasificación de imágenes. Este trabajo propone explorar y desarrollar un sistema de clasificación de imágenes basado en redes neuronales profundas, utilizando como caso de estudio el dataset Fashion-MNIST.

1. TEMA ELEGIDO

El presente proyecto se centra en el estudio y aplicación del Deep Learning, específicamente en el campo de la clasificación de imágenes mediante redes neuronales profundas. La clasificación de imágenes representa uno de los problemas fundamentales en el campo de la visión computarizada y ha experimentado avances significativos gracias a la evolución del aprendizaje profundo.

La clasificación de imágenes mediante Deep Learning constituye un paradigma revolucionario que ha transformado la manera en que las computadoras procesan y comprenden información visual. Este enfoque se distingue de los métodos tradicionales de procesamiento de imágenes por su capacidad de aprender automáticamente características relevantes a partir de los datos, sin necesidad de una extracción manual de características.

2. TIPO DE PROYECTO

El proyecto se clasifica como Tecnológico-Investigativo y se basa en dos dimensiones principales. En la dimensión tecnológica, nos concentramos en implementar y probar un sistema de Deep Learning utilizando frameworks de IA como TensorFlow.

En la dimensión investigativa, realizamos un análisis comparativo de diferentes arquitecturas de redes neuronales preexistentes y estudiamos el impacto de los hiperparámetros en el rendimiento del modelo. Evaluamos empíricamente técnicas de optimización y documentamos los resultados experimentales. Según Chollet (2021), este tipo de proyectos híbridos son esenciales para el avance del campo, combinando rigurosidad científica con aplicación práctica.

3. FUNDAMENTACIÓN TEÓRICA

3.1. Deep Learning: Conceptos Fundamentales

El Deep Learning representa una rama especializada del Machine Learning que se caracteriza por su capacidad para aprender representaciones jerárquicas de datos a través de múltiples niveles de abstracción. Según Goodfellow et al. (2016), el término "profundo" se refiere a la sucesión de capas a través de las cuales los datos son transformados, donde cada capa utiliza la salida de la capa anterior como entrada.

Las arquitecturas de Deep Learning se distinguen por:

- Aprendizaje de Características Automático:

- El sistema aprende automáticamente las características relevantes de los datos
    - Elimina la necesidad de diseñar extractores de características manualmente
    - Permite adaptarse a nuevos tipos de datos sin rediseño
- Representación Jerárquica:
    - Las primeras capas aprenden características básicas
    - Las capas intermedias combinan estas características en patrones más complejos
    - Las capas finales aprenden conceptos abstractos de alto nivel
- Capacidad de Generalización:
    - Habilidad para reconocer patrones en datos nunca antes vistos
    - Robustez ante variaciones en los datos de entrada
    - Adaptabilidad a diferentes dominios de problema

3.2. Deep Learning: Breve reseña histórica

En 1940 Walter Pitts y Warren McCulloch dieron a conocer un estudio de cómo el cerebro humano genera patrones de gran complejidad a través de la interconexión de neuronas. Estos científicos establecieron una comparación entre las células cerebrales y la lógica booleana, creando un modelo informático donde la unidad de procesamiento se basa en algoritmos para imitar el comportamiento neuronal.

En los años 50 Frank Rosenblatt ideó el perceptrón (neurona artificial), más tarde en los años 70 Paul Werbos propuso la retropropagación o backpropagation (se usa para entrenar una red neuronal artificial a través del método regla de la

cadena) y en los 80 Yann LeCun postuló el entrenamiento de algoritmos, en conjunto todos contribuyeron al avance de las redes neuronales.

En el siglo XXI, debido al incremento de la potencia de cómputo y al crecimiento de los datos disponibles (apareciendo la noción de Big Data) el Deep Learning se potenció. Ya a partir de 2010, el deep learning se enriqueció con las llamadas redes neuronales profundas que multiplicaron las aplicaciones. Se optimizó, de este modo, el procesamiento de lenguaje natural, la visión por computadora y el reconocimiento de patrones para identificar imágenes.

3.3. Deep Learning: Avances y aplicaciones del aprendizaje profundo

El aprendizaje profundo ha impulsado avances en diversas áreas. En el campo del reconocimiento de voz, los sistemas de reconocimiento automático del habla basados en aprendizaje profundo han alcanzado niveles de precisión comparables a los humanos. Esto ha permitido aplicaciones como asistentes virtuales, traducción automática de voz y transcripciones precisas.

En cuanto a la visión por computadora el aprendizaje profundo ha sido clave para lograr avances en el reconocimiento y clasificación de objetos en imágenes y videos, como por ejemplo, en la detección de rostros, en la segmentación de imágenes, etc. Las redes neuronales profundas han demostrado su capacidad para comprender y extraer información visual de manera efectiva.

En cuanto al procesamiento del lenguaje natural ha mejorado la capacidad de las máquinas para entender y generar lenguaje humano. Las aplicaciones van

desde chatbots conversacionales hasta sistemas de recomendación de contenido personalizado y análisis de sentimientos en redes sociales.

3.4. Redes Neuronales Artificiales

Las redes neuronales artificiales, fundamentales para el Deep Learning, están inspiradas en el funcionamiento del cerebro humano. Como lo explica Moroney (2021), estas estructuras consisten en unidades de procesamiento interconectadas que operan en paralelo, imitando de esta manera el procesamiento neuronal del cerebro.

Las redes neuronales se componen de varios elementos clave. Las neuronas artificiales son las unidades básicas de procesamiento. Estas neuronas reciben múltiples entradas y generan una única salida, implementando además una función de activación no lineal.

Dentro de la estructura de la red, las capas desempeñan un papel crucial. La capa de entrada recibe los datos originales, mientras que las capas ocultas se encargan de realizar las transformaciones intermedias necesarias. Finalmente, la capa de salida produce el resultado final de la red neuronal. Las conexiones y los pesos son elementos que también son fundamentales en las redes neuronales. Las conexiones son los enlaces entre neuronas, y los pesos asociados a estas conexiones determinan la importancia de cada una. Estos pesos se ajustan durante el proceso de entrenamiento, permitiendo que la red neuronal aprenda y mejore su rendimiento con el tiempo.

3.5. Clasificación de Imágenes

La clasificación de imágenes representa una de las aplicaciones más fundamentales del Deep Learning. Como destaca Krizhevsky et al. (2012), este campo ha experimentado un progreso extraordinario desde la introducción de las arquitecturas profundas.

Las redes neuronales son muy útiles para, a partir de un dato de entrada, hacer una clasificación o regresión a partir de un modelo entrenado. Esta misma idea se puede aplicar a una imagen, para obtener, por ejemplo, qué representa la imagen (clasificación) o detectar un objeto que forme parte de ella (segmentación, detección, identificación, etc).

3.6. Redes Neuronales y TensorFlow

TensorFlow fue desarrollado por Google para satisfacer las necesidades a partir de redes neuronales artificiales. TensorFlow permite construir y entrenar redes neuronales para detectar patrones y razonamientos usados por los humanos. Además de trabajar con redes neuronales, TensorFlow es multiplataforma, trabaja con GPUs y CPUs e incluso con las unidades de procesamiento de tensores (TPUs). TensorFlow es una plataforma que facilita la creación e implementación de modelos de aprendizaje automático. Lo que buscamos hoy en día es automatizar cientos de procesos y TensorFlow nos permite llegar a esta automatización, nos permite crear y entrenar modelos de Aprendizaje Automático con facilidad mediante APIs intuitivas, entre los ejemplos

más comunes que tenemos del uso de TensorFlow es: reconocimiento de imágenes, análisis de sentimientos, diagnósticos médicos, etc.

3.7. Entrenamiento de Redes Neuronales

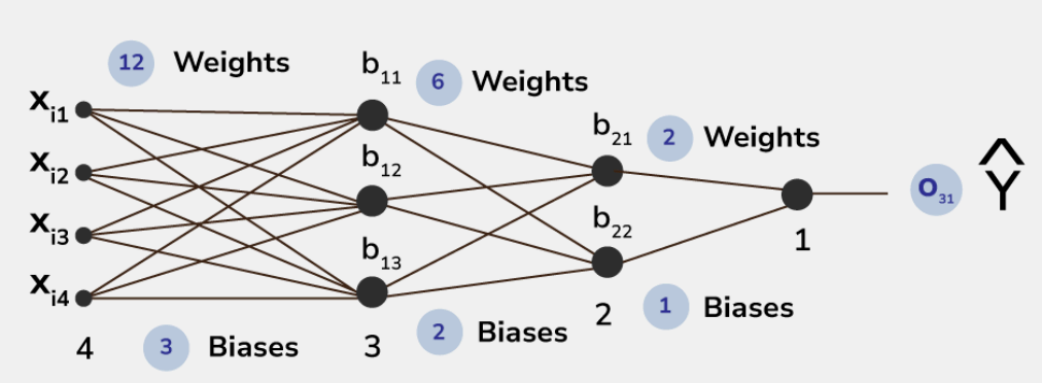
El entrenamiento de una red neuronal es ajustar cada uno de los pesos de las entradas de todas las neuronas que forman parte de una red neuronal, para que las respuestas de la capa de salida se ajusten lo más posible a los datos que conocemos.

El entrenamiento de una red neuronal se basa en cuatro pasos básicos: forward propagation, comparación con el Ground truth (GT), cálculo del error total y backward pass o backpropagation.

3.7.1. Forward propagation

Es el proceso en una red neuronal en el que los datos de entrada pasan a través de las capas de la red para generar una salida. Implica los siguientes pasos:

1. **Capa de entrada:** los datos de entrada se introducen en la capa de entrada de la red neuronal.
2. **Capas ocultas:** los datos de entrada se procesan a través de una o más capas ocultas. Cada neurona de una capa oculta recibe entradas de la capa anterior, aplica una función de activación a la suma ponderada de estas entradas y pasa el resultado a la siguiente capa.
3. **Capa de salida:** los datos procesados pasan por la capa de salida, donde se genera el resultado final de la red. La capa de salida normalmente aplica una función de activación adecuada para la tarea, como softmax para la clasificación o activación lineal para la regresión.
4. **Predicción:** La salida final de la red es el resultado de la predicción o clasificación de los datos de entrada.



3.7.2. Comparación con el Ground truth (GT)

También se conoce como el objetivo para el entrenamiento o la validación del modelo con un conjunto de datos etiquetados. Implica evaluar los resultados de un modelo o sistema en relación con el conjunto de datos que se considera verdadero y preciso. En el contexto de aprendizaje automático y análisis de datos,

esta comparación se usa para medir la precisión, exactitud o desempeño de un modelo.

3.7.3. Cálculo del error total

Consiste en determinar cuánto difieren las predicciones de un modelo respecto a los valores reales o del Ground Truth (GT). Dependiendo del tipo de problema (regresión o clasificación), se utilizan diferentes medidas de error.

3.7.4. Backpropagation

Es un método de cálculo que se utiliza en algoritmos diseñados para entrenar el funcionamiento de redes neuronales artificiales. A grandes rasgos, estas redes neuronales tratan de imitar el funcionamiento de las redes de neuronas biológicas. El método de backpropagation se desarrolla en varias fases:

- Elección del punto de entrada y de salida para el proceso de backpropagation.
- Configurados los valores de entrada y de salida, el algoritmo backpropagation asigna unos valores secundarios que le permitirán modificar parámetros dentro de cada capa y nodo de la red neuronal.
- A partir del análisis de los nodos y capas, se determina el error total (la diferencia entre la salida real y la salida deseada).
- El algoritmo procede a minimizar su efecto en toda la red neuronal.

- El algoritmo backpropagation ajusta los parámetros (pesos y sesgos) para reducir la tasa de error. Se repite el proceso hasta que el error sea mínimo.
- El modelo está listo para hacer una predicción, para lo cual se introducen datos de entrada y el modelo entregará el resultado de salida.

3.8. Accuracy (Precisión)

Nos indica qué tan bien está haciendo su trabajo el modelo al clasificar las imágenes correctamente. Es como un porcentaje de aciertos: si la precisión es del 90%, significa que de cada 100 imágenes, el modelo clasifica correctamente 90. Esta métrica se mide tanto en los datos de entrenamiento (lo que el modelo ya ha visto) como en los de validación o prueba (lo que el modelo no ha visto). Es importante para ver qué tan bien funciona el modelo en general.

3.9. Loss (Pérdida)

Es una medida de qué tan equivocadas están las predicciones del modelo. Es como un puntaje de error: cuanto más baja sea la pérdida, mejor. Cuando el modelo tiene una pérdida baja, significa que está haciendo un buen trabajo ajustando sus predicciones a las respuestas reales. Entender la pérdida es crucial porque te dice si el modelo está aprendiendo bien o si tiene problemas, como el sobreajuste.

3.10. Epochs (Épocas)

Es el número de veces que el modelo pasa por el conjunto completo de datos de entrenamiento. Es un ciclo a través del conjunto de datos de entrenamiento completo, se debe comprender la base y los términos subyacentes que forman un epoch dentro de este contexto. Un epoch se compone de lotes de datos e iteraciones, cuya suma en última instancia equivaldrá a una época. El número de epoch necesarios para que un modelo se ejecute de manera eficiente se basa en los datos en sí y en el objetivo del modelo.

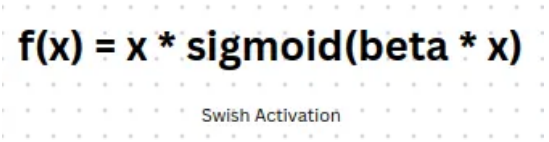
3.11. Overfitting (Sobreajuste)

Es cuando un algoritmo se ajusta demasiado o incluso exactamente a sus datos de entrenamiento, lo que da como resultado un modelo que no puede hacer predicciones o conclusiones precisas a partir de ningún otro dato que no sea el de entrenamiento. Se puede detectar si observamos que la pérdida en los datos de validación empieza a subir mientras que la pérdida en el entrenamiento sigue bajando, lo cual indica que el modelo se está "especializando" demasiado en los datos de entrenamiento.

3.12. Funciones de activación (ReLU, Swish)

La elección de la función de activación puede tener un impacto significativo en el rendimiento de una red neuronal, ya que algunas son más adecuadas para tipos específicos de problemas. Las funciones de activación permiten que las redes neuronales aprendan patrones complejos y matizados en los datos al introducir no linealidad en la red.

- Función Swish (unidad lineal exponencial escalada con un desplazamiento)



Es una fórmula matemática que ayuda a un modelo de aprendizaje profundo a interpretar los datos. Fue inventada por un grupo de investigadores de Google en 2017 y desde entonces ha mostrado resultados prometedores en muchas aplicaciones del mundo real.

- Función ReLU (unidad lineal rectificada)

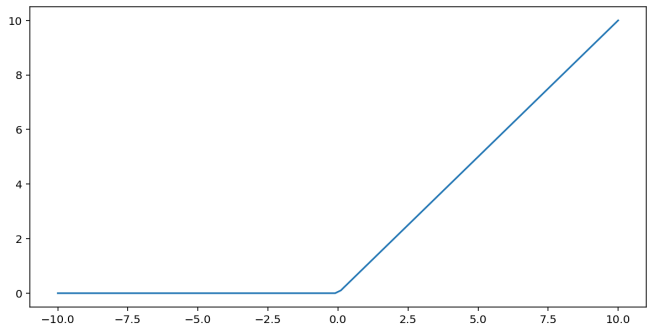
$$F(x) = max(0, x)$$

Transforma el resultado con el siguiente algoritmo:

Si el valor de entrada es menor que 0, muestra 0

Si el valor de entrada es mayor o igual que 0, muestra el valor de entrada.

Gráfico de la función ReLU.



3.13. Dropout

Es una técnica para evitar el sobreajuste. Durante el entrenamiento, "apaga" al azar un porcentaje de las neuronas, lo que obliga al modelo a no depender demasiado de ninguna en particular. Es como si se le pusieran trabas al modelo para que aprenda a hacer su tarea de diferentes maneras. Esto ayuda a que generalice mejor cuando ve datos nuevos.

3.14. Optimizador Adam (Adaptive Moment Estimation)

Es un algoritmo de optimización que combina las ventajas de los algoritmos RMSprop y Momentum para mejorar el proceso de aprendizaje de un modelo. Al igual que Momentum, Adam utiliza una estimación del momento y de la magnitud de los gradientes anteriores para actualizar los parámetros del modelo en cada iteración. Sin embargo, en lugar de utilizar una tasa de aprendizaje constante para todos los parámetros, Adam adapta la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y de la magnitud del gradiente. Esto permite que el modelo se ajuste de manera más eficiente y efectiva a los datos de entrenamiento, lo que puede llevar a una mayor precisión de la predicción en comparación con otros métodos de optimización.

4. VISIÓN DEL PROYECTO

4.1. Objetivo General

Desarrollar e implementar un sistema de clasificación de imágenes basado en Deep Learning que demuestre la capacidad de las redes neuronales profundas para aprender y reconocer patrones visuales complejos.

4.2. Objetivos Específicos

Los objetivos específicos definidos para el proyecto, se detallan a continuación:

1. Analizar y comprender los fundamentos teóricos del Deep Learning y las redes neuronales
2. Implementar y optimizar una arquitectura de red neuronal para la clasificación de imágenes
3. Evaluar y analizar el rendimiento del modelo mediante métricas estándar
4. Documentar y difundir los resultados del desarrollo.

5. SELECCIÓN DE ACCIONES

Esta tabla detalla los objetivos específicos, acciones y habilidades o capacidades requeridas para una investigación en el campo del Deep Learning. El plan de trabajo abarca diversos aspectos fundamentales, como el análisis de los principios teóricos, la comprensión de la arquitectura y funcionamiento de las redes neuronales, y el estudio de técnicas actuales de clasificación de imágenes. Además, incluye la implementación de una arquitectura de red neuronal, la

optimización de los hiperparámetros del modelo, y la medición del rendimiento a través de métricas estándar. El proceso también contempla el análisis del comportamiento del modelo en diferentes escenarios, así como la documentación detallada de todo el desarrollo. Estas actividades y habilidades son esenciales para lograr una sólida formación en el área del Deep Learning y poder aplicar estos conocimientos de manera efectiva en proyectos de investigación y desarrollo.

Objetivo Específico	Acciones	Habilidades o Capacidades
Analizar los principios fundamentales del Deep Learning	Estudiar literatura especializada y realizar pruebas	Comprensión teórica, habilidades analíticas
Comprender la arquitectura y funcionamiento de las redes neuronales	Desarrollar y probar modelos neuronales	Conocimiento técnico, razonamiento lógico
Estudiar las técnicas actuales de clasificación de imágenes	Investigar estudios de casos y aplicar técnicas en proyectos	Capacidad de investigación, análisis crítico
Implementar una arquitectura de red neuronal	Diseñar y codificar la red utilizando frameworks	Programación, conocimiento en frameworks de DL
Optimizar los hiperparámetros del modelo	Realizar pruebas y ajustar parámetros	Habilidades de optimización, atención al detalle
Medir el rendimiento del modelo mediante métricas estándar	Aplicar métricas como precisión, función de costo, entre otros.	Conocimiento de métricas, capacidad de evaluación
Analizar el comportamiento del modelo en diferentes escenarios	Probar el modelo con distintos conjuntos de datos	Flexibilidad, habilidades de análisis de datos
Documentar el proceso de desarrollo	Redactar informes detallados	Habilidades de redacción, documentación técnica
Analizar los resultados obtenidos	Interpretar resultados y redactar informes	Habilidad de interpretación, comunicación clara

6. Fashion-MNIST COMO CASO DE ESTUDIO EN DEEP LEARNING

La selección de Fashion-MNIST proporciona un marco adecuado para explorar los conceptos fundamentales del Deep Learning, manteniendo una conexión directa con aplicaciones prácticas del mundo real. Su complejidad media y sus desafíos representativos lo convierten en una herramienta valiosa tanto para el aprendizaje como para la investigación en visión computarizada y aprendizaje profundo. Este dataset ha sido elegido por varias razones clave.

En primer lugar, logra un balance óptimo entre simplicidad y realismo al representar categorías de ropa y accesorios reconocibles, prácticas para el entrenamiento de redes neuronales. En segundo lugar, ofrece un entorno accesible y eficiente con 70,000 imágenes en escala de grises de 28x28 píxeles, lo cual es ideal para evaluar algoritmos de clasificación. Además, se presenta como una alternativa moderna al clásico MNIST de dígitos escritos a mano, proporcionando un contexto que se asemeja más a aplicaciones reales sin requerir una infraestructura computacional compleja.

7. IMPLEMENTANDO EL CÓDIGO EN COLAB

Comenzamos importando la biblioteca TensorFlow, el cual nos ayudará con el entrenamiento de de modelos de DL y Keras que es una API de alto nivel de TensorFlow para construir y entrenar modelos de aprendizaje profundo.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Cargar el dataset de Fashion-MNIST
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Procedimos a cargar los datos de Fashion-MNIST y dividirlos en dos conjuntos:

- train_images y train_labels: el conjunto de entrenamiento con imágenes y sus etiquetas correspondientes.
- test_images y test_labels: el conjunto de pruebas para evaluar el modelo después del entrenamiento.

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Al cargar el Dataset, lo divide en dos partes:

- x_train y y_train: imágenes y etiquetas para el conjunto de entrenamiento.
- x_test y y_test: imágenes y etiquetas para el conjunto de prueba.

Gracias a los arrays de tres dimensiones x_train y x_test podemos representar un conjunto de imágenes en escala de grises.

x_train, x_test = x_train / 255.0, x_test / 255.0 normaliza los píxeles dividiéndolos por 255, lo que convierte los valores de los píxeles de un rango de 0 a 255 a un rango de 0.0 a 1.0, esto con la finalidad de que el modelo entrene de manera más eficientes debido a redes neuronales suelen converger más rápido cuando las entradas están normalizadas.

Posteriormente se define el modelo y se compila con el optimizador Adam y la función de pérdida de entropía cruzada categórica, y finalmente evalúa el modelo en el conjunto de prueba, mostrando la precisión obtenida.

```
#Incorporación de Dropout al modelo
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Para mejorar la capacidad de generalización del modelo y reducir el sobreajuste se usó una capa Dropout . Durante el entrenamiento, desactiva de forma aleatoria un 20% de las neuronas, evitando que el modelo se vuelva demasiado dependiente de un conjunto específico de ellas. Esto ayuda al modelo

a aprender representaciones más sólidas y a mejorar su capacidad de generalización al trabajar con nuevos datos.

Los resultados obtenidos fueron: accuracy: 0.8743 y loss: 0.3471. En nuestro caso se observa que no era necesario agregar Dropout ya que el rendimiento del modelo disminuyó luego de agregarlo al modelo.

```
#Se agregan neuronas a la capa oculta
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se agregaron neuronas a la capa oculta, al aumentar el número de neuronas en la capa oculta a 512, el modelo puede aprender más y captar patrones más complejos en los datos, sin embargo, esto también implica un riesgo de sobreajuste si el modelo se vuelve demasiado complejo en relación con la cantidad de datos de entrenamiento. Por eso, es fundamental encontrar un equilibrio entre la capacidad del modelo y su habilidad para generalizar bien cuando se enfrenta a datos nuevos. Al agregar neuronas aplicadas a la función relu, no se observa un cambio significativo en la precisión (accuracy).
Los resultados obtenidos fueron: accuracy: 0.8786 y loss: 0.3351. Prácticamente, el rendimiento del modelo no se modifica.

```
#Se cambia función de activación Relu por Elu
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='elu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Reemplazamos la activación ReLU con ELU en la capa oculta. Con este código al realizar el reemplazo, podemos lograr potenciar el rendimiento y la capacidad de aprendizaje frente a la función ReLU.

En nuestro caso el rendimiento del modelo cae en 0,53%. Los resultados obtenidos fueron: accuracy: 0.8753 y loss: 0.3459.

```
#Se modifica la función de activación Relu por Swish
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='swish'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se reemplazó la función de activación Relu por Swish. Con este código al realizar el reemplazo, podemos lograr potenciar el rendimiento y la estabilidad. Al realizar el reemplazo el rendimiento del modelo decae levemente. Los resultados obtenidos fueron: accuracy: 0.8836 - loss: 0.3275

```
#Se modifica la función de activación Relu por Swish y se incrementan neuronas en la capa oculta
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(256, activation='swish'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se agregaron un mayor número de neuronas a la capa oculta y se colocó a Swish como función de activación de dicha capa.

Los resultados obtenidos fueron: accuracy: 0.8763 - loss: 0.3503. El rendimiento es levemente menor que en el modelo original. No es eficiente por el costo computacional y por no mejorar accuracy.

```
#Se agrega una capa más dentro de las capas ocultas
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
       keras.layers.Flatten(input_shape=(28, 28)),
       keras.layers.Dense(128, activation='relu'),
       keras.layers.Dense(64, activation='relu'),  # Nueva capa oculta
       keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se agrega una capa oculta adicional para permitir un aprendizaje más profundo y potencialmente mejorar el rendimiento del modelo, sin embargo, el rendimiento del modelo cae. Los resultados obtenidos fueron: accuracy: 0.8695 - loss: 0.3587.

```
#Se aumenta el número de epochs
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
      keras.layers.Flatten(input_shape=(28, 28)),
      keras.layers.Dense(128, activation='relu'),
      keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se aumenta el número de epochs, al agregar más iteraciones podemos lograr mejorar el aprendizaje, sin embargo, debemos estar atentos a que si se aumenta mucho el número de epochs existe el riesgo de que el modelo aprenda demasiado los detalles del conjunto de entrenamiento y no generalice bien a nuevos datos, lo que se conoce como sobreajuste. Al aumentar los epochs mejoró el rendimiento del modelo. Los resultados obtenidos fueron: accuracy: 0.8803 - loss: 0.3443

```
#Se agrega una capa más de neuronas en la capa oculta y se aumeta epoch
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
      keras.layers.Flatten(input_shape=(28, 28)),
      keras.layers.Dense(128, activation='relu'),
      keras.layers.Dense(64, activation='relu'),
      keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se agregó una capa más de neuronas en la capa oculta y se aumentó el epoch, lo cual generó un aumento de rendimiento. Los resultados obtenidos fueron: accuracy: 0.8846 - loss: 0.3316

```
#Se agrega una capa más de neuronas en la capa oculta (3 capas)
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
       keras.layers.Flatten(input_shape=(28, 28)),
       keras.layers.Dense(128, activation='relu'),
       keras.layers.Dense(64, activation='relu'),
       keras.layers.Dense(64, activation='relu'),
       keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=20)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se agregó una capa más de neuronas en la capa oculta, implementando un modelo de red neuronal con tres capas ocultas, y aumentó el rendimiento. Los resultados obtenidos fueron: accuracy: 0.8894 - loss: 0.3692.

```
[ ]  #Se agrega cantidad de  neuronas en la capa oculta 2
     (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
     x_train, x_test = x_train / 255.0, x_test / 255.0

     model = tf.keras.models.Sequential([
            keras.layers.Flatten(input_shape=(28, 28)),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(10, activation='softmax')
        ])

     model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

     model.fit(x_train, y_train, epochs=20)

     test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
     print('\nTest accuracy:', test_acc)
```

Se agregan neuronas en la capa oculta 2 de un esquema de 3 capas ocultas. Los resultados obtenidos fueron: accuracy: 0.8940 - loss: 0.3499.

```
#Se agrega una capa oculta más y se aumenta epoch a 25
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
       keras.layers.Flatten(input_shape=(28, 28)),
       keras.layers.Dense(128, activation='relu'),
       keras.layers.Dense(128, activation='relu'),
       keras.layers.Dense(64, activation='relu'),
       keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=25)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se opera con 3 capas ocultas y se aumenta el epoch a 25. Los resultados obtenidos fueron: accuracy: 0.8876 - loss: 0.3995

```
#Se agrega otra capa oculta
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
       keras.layers.Flatten(input_shape=(28, 28)),
       keras.layers.Dense(128, activation='relu'),
       keras.layers.Dense(96, activation='relu'),
       keras.layers.Dense(32, activation='relu'),
       keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=20)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se agrega otra capa oculta.

```
#se cambia el optimizador
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=20)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Se cambia el optimizador utilizado en el modelo de red neuronal de Adam a SGD (Stochastic Gradient Descent). Los resultados obtenidos fueron: accuracy: 0.8691 - loss: 0.3708

```
# 1. Cargar los datos
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# 2. Normalizar los datos (convertir los valores de píxeles de 0-255 a 0-1)
x_train, x_test = x_train / 255.0, x_test / 255.0

# 3. Crear el modelo - una red neuronal simple pero efectiva
modelo = keras.Sequential([
    # Aplanar la imagen
    keras.layers.Flatten(input_shape=(28, 28)),

    # Primera capa oculta con más neuronas
    keras.layers.Dense(256, activation='relu'),
    # Agregar Dropout para evitar sobreajuste
    keras.layers.Dropout(0.2),

    # Segunda capa oculta
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2),

    # Capa de salida (10 clases para los diferentes tipos de ropa)
    keras.layers.Dense(10, activation='softmax')
])

# 4. Compilar el modelo
modelo.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# 5. Entrenar el modelo
# Aumentamos el número de epochs y agregamos validation_split
historia = modelo.fit(
    x_train,
    y_train,
    epochs=15,                # Más epochs para mejor aprendizaje
    batch_size=32,           # Tamaño del batch pequeño
    validation_split=0.2     # Usar 20% de los datos para validación
)

# 6. Evaluar el modelo
test_loss, test_acc = modelo.evaluate(x_test, y_test, verbose=1)
print('\nPrecisión en el test:', test_acc)
```

Y acá mostramos en 6 pasos cómo construimos, entrenamos y evaluamos un modelo de red neuronal simple para clasificar imágenes de ropa del dataset Fashion-MNIST:

- ➔ Cargar los datos: Cargamos el dataset y dividimos en conjuntos de entrenamiento.
- ➔ Normalizamos los datos: Se convirtieron los píxeles de 0 a 255 a un rango de 0 a 1.
- ➔ Se creó el modelo: Con una red neuronal simple pero efectiva, se aplanó cada imagen de 28x28 pixeles en vector de 784 elementos para que la red neuronal pueda procesarlo.
- ➔ Compilamos el modelo: Para el optimizador nos decidimos por Adam.
- ➔ Entrenamos el modelo: Aumentamos el número de epochs y agregamos validation_split para monitorear el rendimiento del modelo en datos no vistos durante el entrenamiento.
- ➔ Evaluamos el modelo: Evaluamos en datos nuevos para verificar su rendimiento.

```python
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
       keras.layers.Flatten(input_shape=(28, 28)),
       keras.layers.Dense(256, activation='swish'),
       keras.layers.Dropout(0.2),
       keras.layers.Dense(128, activation='relu'),
       keras.layers.Dropout(0.2),
       keras.layers.Dense(10, activation='softmax')
   ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=15)

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nTest accuracy:', test_acc)
```

Este es el modelo con mejor rendimiento y función de costo más baja. Los resultados obtenidos fueron: accuracy: 0.8923 - loss: 0.3224.

El modelo de red neuronal incluye:

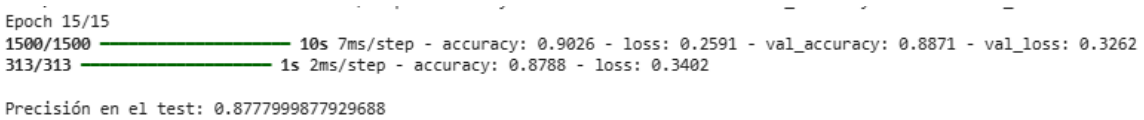
- ➔ Dos capas ocultas con 256 y 128 neuronas.
- ➔ Una combinación de funciones de activación Swish y ReLU.
- ➔ Capas de Dropout para reducir el sobreajuste.
- ➔ Entrenamiento con 15 epochs para un aprendizaje más profundo.
- ➔ Evaluación final para medir la precisión en datos de prueba

8. RESULTADOS

El modelo comienza con una precisión de entrenamiento de 0.7424 (74.24%) y una precisión de validación 0.8488 (84.88%):

```
Epoch 1/15
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.7424 - loss: 0.7177 - val_accuracy: 0.8499 - val_loss: 0.4094
Epoch 2/15
```

Indicándonos que la pérdida es alta al principio, sin embargo, se puede tomar como algo normal ya que el modelo está empezando a aprender. A medida de que se va avanzando la precisión de entrenamiento va mejorando de forma consistente llegando a 90.26% indicándonos que el modelo está captando bien los patrones del conjunto de entrenamiento y obteniendo una precisión final de aproximadamente 87.88%.



El modelo logró una precisión cercana al 91% en los datos de entrenamiento y alrededor del 89% en los datos de prueba, lo que muestra un buen equilibrio entre aprender de los datos y poder generalizar a datos nuevos. Aunque la pérdida en el conjunto de prueba es un poco más alta que en el entrenamiento, no parece haber un sobreajuste preocupante, con esto concluimos que el modelo es apto para clasificar imágenes nuevas con un rendimiento confiable.

9. REFERENCIAS BIBLIOGRÁFICAS

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Moroney, L. (2021). AI and Machine Learning for Coders: A Programmer's Guide to Artificial Intelligence. O'Reilly Media.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature, 521(7553), 436-444.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, 25, 1097-1105.

Chollet, F. (2021). Deep Learning with Python. Manning Publications.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into Deep Learning. arXiv preprint arXiv:2106.11342.

Howard, J., & Gugger, S. (2020). Deep Learning for Coders with fastai and PyTorch. O'Reilly Media.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770-778.

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going Deeper with Convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1-9.

Chollet, F. (2021). *Deep Learning with Python* (2nd ed.). Manning Publications.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature*, 521(7553), 436-444.

IDECOR. (2019, August 28). ¿Deep Learning… y clasificación de imágenes? IDECOR. https://www.idecor.gob.ar/introduccion-a-las-redes-neuronales-convolucionales-para-clasificacion-de-imagenes/

Deep Learning: clasificando imágenes con redes neuronales. (2020, May 25). LIS Data Solutions. https://www.lisdatasolutions.com/es/blog/deep-learning-clasificando-imagenes-con-redes-neuronales/

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60, 84–90.

Follow Improve. (2024, May 1). What is Forward Propagation in Neural Networks? GeeksforGeeks. https://www.geeksforgeeks.org/what-is-forward-propagation-in-neural-networks/

Adam. (n.d.). Interactivechaos.com. Retrieved November 8, 2024, from https://interactivechaos.com/es/manual/tutorial-de-machine-learning/adam

¿Qué es el sobreajuste? (2023, May 16). Ibm.com. https://www.ibm.com/es-es/topics/overfitting