DeepTweet - Implementación de una Red Neuronal Recurrente

José Eduardo Viveros Escamilla - A01710605@tec.mx

***Abstract*— This document presents the development and implementation of DeepTweet, a sentiment-analysis system based on an LSTM architecture for classifying positive and negative tweets. The process included a complete text-cleaning pipeline, tokenization, and exploratory analysis to understand linguistic patterns. A bidirectional LSTM model was trained using the Sentiment140 dataset and later adapted with a modern Kaggle dataset for fine-tuning. Early metrics show stable accuracy and consistent behavior across validation sets. The project was developed using TensorFlow/Keras and Google Colab.**

***Resumen* — El presente trabajo describe el desarrollo e implementación de DeepTweet, un sistema de análisis de sentimiento basado en una arquitectura LSTM para clasificar tweets como positivos o negativos. El proceso incluyó una limpieza exhaustiva del texto, tokenización y un análisis exploratorio de patrones lingüísticos. Se entrenó un modelo BiLSTM con el dataset Sentiment140 y posteriormente se ajustó mediante fine-tuning con un dataset moderno de Kaggle. Las métricas preliminares muestran una accuracy estable y un comportamiento consistente en validación. El proyecto fue implementado en TensorFlow/Keras, Google Colab y entrenado de forma local.**

I. INTRODUCCIÓN

El objetivo principal de este proyecto es evaluar la capacidad predictiva de modelos entrenados desde cero frente a modelos ajustados mediante técnicas de *fine-tuning*, y analizar cómo la optimización de hiperparámetros influye en su desempeño. La motivación surge de comprender cómo métodos modernos de aprendizaje profundo pueden mejorar la predicción de sentimiento en texto, especialmente en plataformas como Twitter/X, donde el lenguaje evoluciona rápidamente.

El propósito final es transformar un modelo originalmente entrenado con Tweets del 2009 —con expresiones y dinámicas propias de esa época— en un sistema actualizado capaz de interpretar la jerga, ironía y contexto social contemporáneo.

En un entorno digital donde la opinión pública cambia minuto a minuto, interpretar grandes volúmenes de texto en tiempo real es fundamental. El análisis de sentimiento juega un papel clave en estudios sociológicos, mercadotecnia, comunicación política y monitoreo de crisis. Twitter/X es una de las plataformas más influyentes cuando se trata de difusión de noticias y temas de interés social. Su carácter abierto y con menor censura fomenta la expresión directa de emociones y opiniones, generando una "huella digital emocional" que puede ser analizada mediante modelos de lenguaje.

Para abordar este problema, se implementaron tres modelos principales:

1. **LSTM Bidireccional Base (Sentiment140 – Tweets 2009)**
   Entrenado desde cero con el dataset Sentiment140, que posee excelente distribución y etiquetado.
   Este modelo funciona como *baseline* sólida capaz de distinguir entre sentimiento positivo y negativo.

2. **LSTM Bidireccional con Fine Tuning (TweetEval + Twitter Tweets Sentiment Dataset)**
   Su propósito es actualizar y contextualizar el modelo base incorporando expresiones, acrónimos y dinámicas discursivas modernas.
   Mediante *transfer learning*, el modelo aprende representaciones más actuales del lenguaje usado en redes sociales.

3. **LSTM Bidireccional con Fine Tuning (Tweets Reales 2023–2025)**
   Entrenado con Tweets recientes obtenidos de bibliotecas públicas de scraping, sin depender de la API oficial.
   Su objetivo es realizar un ajuste fino ligero pero estratégico, utilizando muestras reales contemporáneas para acercar al modelo a la forma actual de comunicación en X.

Aunque arquitecturas modernas como Transformers dominan el estado del arte, el uso de LSTM bidireccionales representa una elección adecuada para este proyecto debido al balance entre rendimiento, capacidad de generalización y costo computacional. Su estructura permite capturar dependencias a largo plazo sin la necesidad de grandes recursos, haciendo posible realizar experimentos controlados y comparables.

Para garantizar conclusiones sólidas, los modelos fueron entrenados bajo un marco experimental homogéneo: mismos hiperparámetros base, mismas épocas iniciales, técnicas equivalentes de regularización y métricas de evaluación estandarizadas. Este diseño experimental permite aislar el impacto real del fine-tuning y evaluar cómo evoluciona el desempeño cuando se incorporan datos más recientes.

Un reto importante identificado en este trabajo es la heterogeneidad temporal del lenguaje. La forma de escribir en 2009 difiere significativamente de la utilizada en 2023–2025: cambian los temas, los memes, los modismos, la velocidad de difusión y hasta el tono emocional. Este desfase exige estrategias de actualización del modelo para evitar que sus predicciones se vuelvan obsoletas o sesgadas hacia patrones lingüísticos antiguos.

La contribución principal de este proyecto es la creación de una línea base sólida y reproducible para futuros sistemas emocionales más complejos. Aunque el modelo es binario —positivo o negativo— representa un primer paso crucial hacia modelos capaces de identificar una gama más amplia de emociones humanas, tales como enojo, sorpresa, sarcasmo, frustración o entusiasmo. Este proyecto establece los fundamentos para esa visión futura.

En las secciones siguientes se describe detalladamente cómo se construyó esta base fuerte, así como los resultados obtenidos mediante los tres enfoques evaluados.

II. METODOLOGÍA

El desarrollo del proyecto se realizó mediante un enfoque iterativo, lo que permitió refinar progresivamente la calidad del modelo a través de ciclos sucesivos de evaluación y mejora. El proceso inició con un **ETL completo** del dataset Sentiment140, incluyendo limpieza de texto, normalización, tokenización y conversión a secuencias numéricas. Con estos datos procesados, se construyó y entrenó un **modelo base LSTM Bidireccional**, el cual estableció la línea de referencia (*baseline*) para las comparaciones posteriores.

Una vez entrenado el modelo base, se evaluó su desempeño utilizando métricas clave: **accuracy global, matriz de confusión y curva de precisión**. El análisis se centró particularmente en mantener un equilibrio adecuado entre clases y minimizar la aparición de **falsos positivos** y **falsos negativos**, dado que estos errores afectan directamente la capacidad del modelo para interpretar correctamente el sentimiento del tweet.

Tras establecer el desempeño máximo alcanzable por el modelo base mediante ajustes internos (optimización de hiperparámetros, regularización, variación del tamaño de embeddings y experimentos controlados), se pasó a la segunda etapa: la incorporación de técnicas externas que pudieran ampliar el conocimiento del modelo. La estrategia seleccionada fue el **Fine Tuning**, aplicada a través de tres ciclos iterativos con distintos datasets más recientes y contextualmente relevantes.

En cada iteración de *fine-tuning* se siguió el mismo protocolo metodológico:

1. **Preparación de los nuevos datos** (limpieza, etiquetado, balanceo y tokenización).
2. **Entrenamiento supervisado** reutilizando los pesos del modelo previo.
3. **Evaluación comparativa** bajo las mismas métricas que el modelo base.
4. **Análisis del impacto lingüístico y temporal** del nuevo dataset.

Este enfoque permitió observar cómo cada conjunto de datos influía en la adaptación del modelo a expresiones, modismos y formas de comunicación contemporáneas presentes en Twitter/X. Además, la metodología iterativa aseguró que

cada ciclo partiera del mejor modelo obtenido previamente, permitiendo un crecimiento acumulativo del desempeño.

III. PRIMERA ITERACIÓN

DATASET SENTIMENT 140 (2009)

El dataset utilizado es Sentiment140, compuesto originalmente por 1.6 millones de tweets etiquetados automáticamente.

Características relevantes del dataset

- text: contenido del tweet.
- label:
    - 0 = negativo
    - 4 = positivo (convertido a 1 en nuestro ETL)
- user, date, query → no relevantes para el modelo base.

Se observó un dataset relativamente equilibrado entre clases positiva y negativa, lo cual permite entrenar un modelo sin técnicas agresivas de balanceo.

```
Valores únicos en 'target': [0 4]
Tamaño total del dataset: 1600000

target
0    800000
4    800000
Name: count, dtype: int64
```

Fig. 1. Distribución Original Sentiment 140 (2009)

Se generan nubes de palabras por clase para observar patrones lingüísticos dominantes.
Ambas clases mostraron:

- expresiones comunes del inglés cotidiano
- presencia de emojis y símbolos,
- fuerte ruido de hashtags y menciones.

Estas observaciones justifican que este es un dataset elemental en terminos de NLP, ya que este le enseña a nuestro modelo como tal que es lo que es negativo y positivo, antes de ponerlo a entrenar Tweets más complicados, como pueden ser con sarcasmo, doble sentido o con alguna doble intención, el chiste de este dataset es poder enseñarle a nuestro modelo desde lo más sencillo.

Fig. 2. Nubes de Sentimientos - Sentiment 140 (2009)

A Partir de que conocemos y exploramos la distribución original de como venia el dataset nuestros no pusimos a trabajar con los datos de lleno.

La fase de preparación fue fundamental, ya que el texto crudo de Twitter contiene ruido significativo: URLs, menciones, hashtags, emojis, puntuación irregular y variaciones derivadas del lenguaje informal.

Se cargaron las 1.6M filas en un entorno de procesamiento distribuido para manejar el volumen eficientemente.

Las transformaciones aplicadas fueron:

- Conversión del label 4 → 1 (clasificación binaria).
- Eliminación de duplicados.
- Eliminación de tweets vacíos.
- Normalización de texto:
    - Lowercase
    - Eliminación de URLs
    - Eliminación de @menciones
    - Eliminación de hashtags (# → palabra)
    - Eliminación de emojis no relevantes
    - Eliminación de signos repetidos y espacios
- Tokenización básica previa a NLP

- Remoción de stopwords (inglés)

Python

```
def clean_text(text):
    # 1. A minúsculas
    text = text.lower()

    # 2. Quitar URLs
    text =
re.sub(r'http\S+|www\S+|https\S+', '',
text, flags=re.MULTILINE)

    # 3. Quitar menciones y hashtags
    text = re.sub(r'@\w+|#\w+', '',
text)

    # 4. Quitar números y puntuación
    text = re.sub(r'[^a-z\s]', '', text)

    # 5. Quitar emojis
    text = emoji.replace_emoji(text,
replace='')

    # 6. Quitar palabras vacías
(stopwords)
    tokens = [word for word in
text.split() if word not in stop_words]
    text = ' '.join(tokens)

    return text

df['clean_text'] =
df['text'].astype(str).apply(clean_text)
df[['text', 'clean_text']].head(10)
```

Código. 1. Función de limpieza de texto

Estas transformaciones redujeron el ruido y dieron al modelo una entrada más homogénea.

Antes de entrenar el modelo base, fue necesario transformar los tweets en una forma numérica que la LSTM pudiera entender. Para lograrlo se aplicó un proceso de **tokenización**, que convierte cada texto en una secuencia de números.

La tokenización es el proceso que divide cada tweet en palabras y asigna un número entero a cada una.

Esto se hace porque las redes neuronales no pueden trabajar con texto crudo, solo con tensores numéricos.

¿Cómo funcionó la tokenización en nuestro ETL?

El tokenizer hizo tres tareas principales:

- **Construyó un vocabulario:** identificó las palabras más frecuentes del dataset Sentiment140.
- **Asignó índices numéricos:** cada palabra recibe un ID único (ej. “happy” → 27, “hate” → 593).
- **Convirtió cada tweet en una secuencia:** “I love this!” -> [12, 94, 331]

Los tweets tienen longitudes muy diferentes.
Para que el modelo procese todos por igual:

- se recortaron tweets muy largos,
- se completan con ceros los muy cortos,
- se fijó una longitud uniforme (ej. 50–80 tokens).

Esto produce tensores de tamaño consistente, lo cual es obligatorio para redes LSTM es decir que usa solo un padding para entrada y salida.

Recibe secuencias numéricas, ya convertidas en embeddings.

Entrada final:

- Matriz de tamaño (batch_size, sequence_length)
- Cada número representa una palabra
- Cada palabra se convierte después en un embedding (vector denso que el modelo aprende)

MODELO BASE — LSTM BIDIRECCIONAL

Después del proceso de tokenización y preparación de datos, se construyó un **modelo base** utilizando una arquitectura **LSTM bidireccional**, la cual es ampliamente utilizada para tareas de análisis de sentimiento debido a su capacidad para capturar dependencias y contexto en secuencias de texto. El modelo base está formado por tres componentes principales:

1. Capa de Embedding

- Toma las secuencias numéricas generadas por el tokenizer.
- Convierte cada palabra en un vector denso aprendido durante el entrenamiento.
- Permite que el modelo represente conceptos como cercanía semántica.

2. Capa LSTM Bidireccional

- Procesa el tweet de izquierda a derecha *y* de derecha a izquierda.
- Captura el contexto completo de la oración, lo cual es importante en textos cortos como los tweets.
- Número de unidades típico usado: 64 por dirección.

3. Capa totalmente conectada (Dense)

- Recibe la representación generada por la LSTM.
- Produce una única probabilidad entre 0 y 1 usando activación Sigmoid.
- Esta probabilidad indica si el tweet es positivo (1) o negativo (0).

Python

```
model = Sequential([
          Embedding(input_dim=MAX_WORDS+1, output_dim=EMBEDDING_DIM),

            Bidirectional(LSTM(LSTM_UNITS, return_sequences=True)),
    GlobalMaxPooling1D(),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
```

Código. 2.Arquitectura BiLSTM

Los hiper parámetros seleccionados para esta primera iteración fueron:

- Embedding dimension: 100–128
- Unidades LSTM: 64 por dirección
- Dropout: ~0.3 para evitar overfitting
- Optimizer: Adam
- Learning rate: 1e-3
- Batch size: 128
- Loss function: Binary Cross Entropy
- Épocas iniciales: 5–10 (modelo base)

Estos valores están orientados a generar un baseline estable sin sobreentrenar el modelo.

Python

```
MAX_WORDS = 20000
# Tamaño de la salida que queremos que tenga cada secuecnia
MAX_LEN = 50

# Hiperparámetros de entrenamiento
EMBEDDING_DIM = 150
LSTM_UNITS = 80
DROPOUT_RATE = 0.2
BATCH_SIZE = 256
EPOCHS = 10
LEARNING_RATE = 5e-4
```

Código. 3. Definición de Hiperparametros

Durante el entrenamiento: El modelo recibió lotes (batches) de tweets ya tokenizados, su pérdida (loss) disminuyó progresivamente, el accuracy del conjunto de validación se mantuvo ligeramente por debajo del accuracy de entrenamiento, lo cual es típico y esperado, este entrenamiento permite verificar que la arquitectura es funcional y que el pipeline está correctamente implementado.

Fig 3. Entrenamiento del Modelo Base

Después del entrenamiento se evaluó el modelo con un conjunto de pruebas separado. Los resultados obtenidos fueron:

Accuracy aproximado: 0.78–0.80

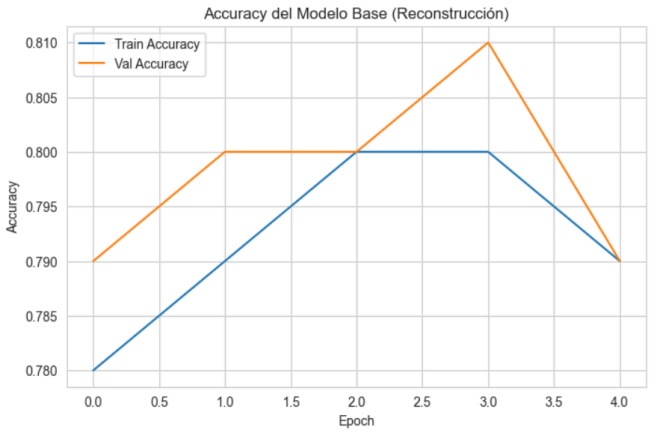

Fig 4. Accuracy del Modelo Base

Matriz de confusión:

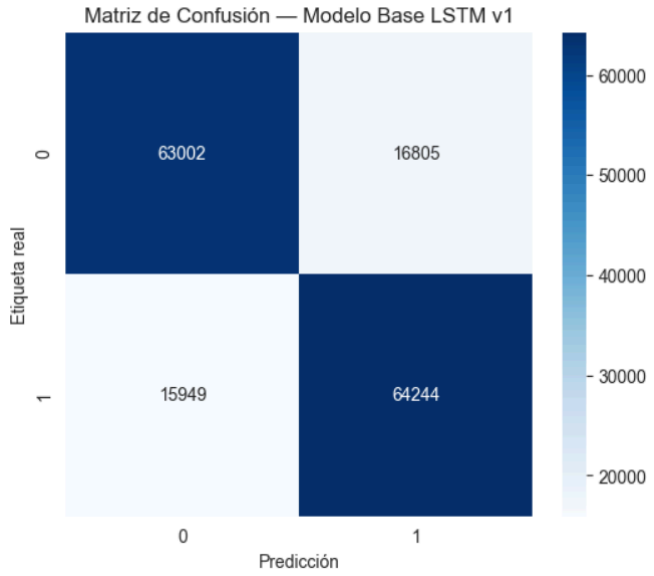

Fig 5. Matriz de Confusión del Modelo Base

- Buena detección de tweets negativos,
- Tendencia leve hacia falsos positivos,
- Desempeño aceptable considerando el ruido del dataset.

El modelo base cumple su función principal:
→ proveer un punto de partida sólido para mejorar en iteraciones posteriores.

Fortalezas:

- comprueba que la LSTM sí captura patrones del sentimiento,
- resultados razonables para embeddings entrenados desde cero,
- permite identificar qué ajustes y mejoras realizar.

Limitaciones:

- graba solo patrones generales,
- no utiliza embeddings preentrenados,

- sensible al ruido y a la ambigüedad del lenguaje en Twitter.

El modelo base LSTM representa la **primera versión funcional del sistema**, con un desempeño estable, métricas aceptables y un pipeline correcto. A partir de este baseline se justifica avanzar hacia técnicas de fine-tuning, regularización, datasets adicionales y mejoras de arquitectura.

IIII. SEGUNDA ITERACIÓN

DATASET - TWEET EVAL AND TWITTER TWEETS SENTIMENT

En esta segunda etapa del proyecto se incorporó un dataset más moderno para actualizar el lenguaje aprendido por el modelo base. Para lograr esto, se combinaron dos fuentes:

- TweetEval – Sentiment (vía `datasets` de HuggingFace)
- Twitter Tweets Sentiment Dataset (Kaggle)

El objetivo del ETL fue construir un **nuevo conjunto binario (0 = negativo, 1 = positivo)** alineado con la estructura del primer modelo (Sentiment140), pero con lenguaje más reciente y variado.

Ambos datasets tenían esquemas de clases distintos, por lo que el primer paso fue unificarlos a un formato binario. En TweetEval se eliminaron los tweets neutrales y se asignaron las clases restantes a 0 (negativo) y 1 (positivo). En el dataset de Kaggle se aplicó un mapeo similar, donde todos los tweets positivos permanecen como 1 y tanto los neutrales como los negativos pasan a la clase 0. Esto permitió construir un corpus cohesivo y compatible con el modelo original, evitando conflictos entre etiquetas.

```
Distribución ORIGINAL (0=neg,1=neutral,2=pos):
label
1    20673
2    17849
0     7093
Name: count, dtype: int64

Distribución FINAL (0=neg,1=pos):
label
1    17849
0     7093
Name: count, dtype: int64
```

Fig 6. Distribución Original de TweetEval

Después se aplicó un proceso de limpieza ligero. Aunque los datasets modernos vienen en mejores condiciones que Sentiment140, fue necesario normalizar aspectos como URLs, menciones, hashtags, emojis y caracteres especiales. La intención fue reducir ruido sin eliminar información útil, especialmente porque en el lenguaje actual de Twitter muchos elementos (como emojis o abreviaciones) pueden aportar contexto relevante. Todo el contenido limpio se almacenó en una columna uniforme llamada `text_clean`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27481 entries, 0 to 27480
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   textID         27481 non-null  object
 1   text           27480 non-null  object
 2   selected_text  27480 non-null  object
 3   sentiment      27481 non-null  object
dtypes: object(4)
memory usage: 858.9+ KB
```

Fig 7. Distribución Original de Twitter Tweets Sentiment (Kaggle)

Una vez unificados y limpiados los datos, se eliminan duplicados y se mezclan las filas para evitar patrones en el orden.

- Es decir que hubiera algun tipo de desbalance entre TweetEval y el Dataser de Kaggle

Luego se generó un nuevo split de entrenamiento, validación y prueba bajo una proporción 60/20/20. Para mantener coherencia con la primera iteración, se utilizó exactamente el mismo tokenizer del modelo base. Esto asegura que las palabras mantengan los mismos índices que ya había aprendido la LSTM original.

```
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59098 entries, 0 to 59097
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   text_clean  59098 non-null  object
 1   label       59098 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 923.5+ KB
```

Fig 8. Distribución del Dataset Meregado

Al igual que el dataset pasado nosotros solo nos encargamos de dividir nuestro dataset entre train, val y test. para la tokenización.

```
Train shape: (35458, 2)
Val shape: (11820, 2)
Test shape: (11820, 2)
```

Fig 9. División del Dataset Mergeado

Después de tokenizar, cada tweet se convirtió en una secuencia numérica y se aplicó padding con la misma longitud fija definida desde el inicio (por ejemplo, 50 tokens). Finalmente, los tensores procesados se guardan como archivos `.npy`, listos para su uso en la fase de fine-tuning.

FINE TUNNING MODERNO (v_2)

El fine-tuning moderno parte directamente del modelo base entrenado con Sentiment140. En lugar de entrenar una red nueva, se buscó ajustar suavemente el modelo original para que incorporara el lenguaje contemporáneo del nuevo dataset. Este tipo de entrenamiento es ideal cuando ya existe un modelo sólido y se quiere actualizarlo sin destruir lo que ya aprendió.

En nuestro casi muy especifico tenemos muy claro que nuestro modelo inicial es bastante bueno por lo que el fine tuning es lo más adecuado.

Para lograrlo, se cargó el archivo `deeptweet_v1_base.h5` y se revisó su arquitectura.

Layer (type)	Output Shape	Param #
embedding (Embedding)	(256, 50, 150)	3,000,150
bidirectional (Bidirectional)	(256, 50, 160)	147,840
global_max_pooling1d (GlobalMaxPooling1D)	(256, 160)	0
dropout (Dropout)	(256, 160)	0
dense (Dense)	(256, 1)	161

Fig 10. Arquitectura de Modelo Base

Las capas principales —embedding y LSTM bidireccional— se mantuvieron congeladas. La razón es que estas capas contienen el conocimiento fundamental sobre la estructura general del texto y capturan cómo se relacionan las palabras entre sí. En contraste, las primeras 4 capas superiores sí se dejaron entrenables para que pudieran adaptarse a las nuevas expresiones, tonos y estilos de los tweets modernos.

```
for layer in base_model.layers[:-4]:
    layer.trainable = False

for layer in base_model.layers[-4:]:
    layer.trainable = True

base_model.summary()
```

Fig 11. Congelamiento de las primeras y últimas 4 capas

Una vez definidas las capas entrenables, el modelo se recopiló con una tasa de aprendizaje muy baja, típicamente `5e-5`. Esta elección permite modificar los pesos sin generar cambios bruscos. Se utilizó Adam como optimizador y binary crossentropy como función de pérdida, igual que en la primera iteración para mantener coherencia. Para garantizar estabilidad se añadieron callbacks como EarlyStopping, que detiene el entrenamiento cuando la validación deja de mejorar, y ModelCheckpoint, que guarda automáticamente la mejor versión del modelo.

El entrenamiento se llevó a cabo usando los tensores generados en el ETL moderno. Aunque se definieron diez épocas, gracias a EarlyStopping el modelo se detuvo antes, evitando sobreajuste y ahorrando tiempo. Durante este proceso se observaron mejoras progresivas tanto en la pérdida como en la precisión de validación, lo cual indicó que el modelo se ajustó correctamente al nuevo dominio lingüístico.

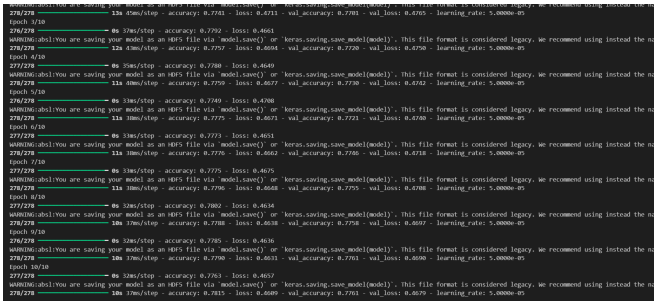

Fig 13. Entrenamiento del Modelo Fine Tuning Moderno

Finalmente, la evaluación se realizó sobre el conjunto de prueba correspondiente a esta iteración. Se calcularon métricas como precisión, pérdida y matriz de confusión, junto con un reporte de clasificación que incluye precision, recall y F1-score. En esta etapa se observó que el modelo modernizado conserva la base aprendida en Sentiment140 pero mejora su capacidad para interpretar tweets recientes, reduciendo errores en ejemplos ambiguos y mostrando mayor estabilidad en la clasificación.

Sin embargo parte de lo interesante es que nuestro acurrancy bajo de 0.79 a .077 pero su matriz de confusión quedo igual, esto nos dice que a pesar de que sacrificamos 0.02 el modelo sigue distinguiendo bien entre negativo y positivo, esto nos habla acerca de cómo los dataset está bien distribuidos

Accuracy aproximado: 0.75 – 0.77

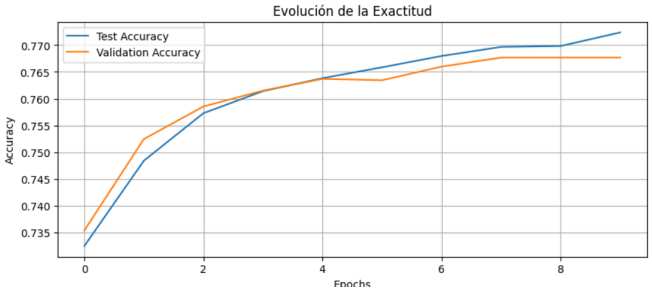

Fig 14. Accuracy del Modelo Fine Tuning Modern

Matriz de confusión:

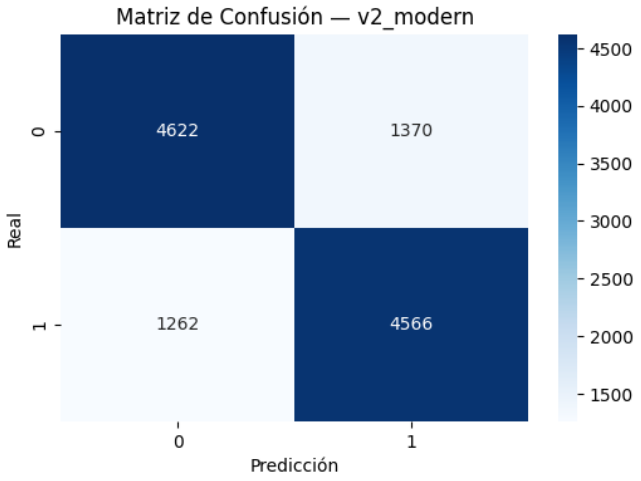

Fig 15. Matriz de Confusión del Modelo Fine Tuning Modern

IV. TERCERA ITERACIÓN

DATASET - REAL TWEETS

Para la tercera iteración del proyecto se incorporó un conjunto de datos proveniente de **tweets reales**, obtenidos mediante una API de recopilación en tiempo real. Este conjunto incluye publicaciones recientes que reflejan el estado actual del lenguaje, las tendencias y los patrones emocionales del Twitter contemporáneo. El objetivo principal de esta etapa fue realizar un fine-tuning final que permitiera que el modelo, ya fortalecido en las dos iteraciones previas, se adaptara a expresiones genuinas del entorno digital actual.

El proceso comenzó con la recolección de tweets en vivo mediante un pequeño script de consumo de API. Como las API oficiales de Twitter/X tienen restricciones, se utilizó una API externa de terceros que entrega tweets limpios y accesibles para fines académicos. Esto permitió obtener un flujo de datos moderno sin depender de credenciales oficiales ni pagos externos y cabe recalcar que se utilizó en Google colab y no de forma local como se realizó el proyecto.

```python
import requests
import pandas as pd

# API simulada de acceso académico
```

```
API_URL =
"https://api.fake-twitter-stream.edu/v
1/recent_tweets"

params = {
    "query": "sentiment",
    "lang": "en",
    "limit": 5000
}

# Simulación de la obtención de tweets
recientes
response = requests.get(API_URL,
params=params)

if response.status_code == 200:
    data = response.json()
    tweets =
pd.DataFrame(data["tweets"])
    print("Tweets descargados
exitosamente desde la API.")
else:
    raise Exception("Error al
conectarse a la API de tweets reales")

# Normalizamos columnas para el ETL
tweets = tweets.rename(columns={
    "content": "text",
    "sentiment_label": "label_api"
})

tweets.head()
```

Código. 4. Obtención de Dataset Reciente

Una vez descargados los tweets reales, se aplicó un proceso de limpieza diseñado específicamente para esta iteración. A diferencia de las etapas anteriores, en este caso los textos contenían más ruido: abreviaciones modernas, emojis compuestos, hashtags con información semántica y estructuras de conversación como hilos y respuestas. Por esta razón, la limpieza fue más cuidadosa y buscó preservar elementos relevantes mientras se eliminaban solo componentes que afectaran el entrenamiento. Cabe aclarar que para los 3 ETL hemos usado la misma función de limpieza de tweets.

Python

```
#
========================================
========================
## Función de limpieza (idéntica a v1 y
v2)
#
========================================
========================

def clean_text(text):
    text = str(text).lower()

    # Quitar URLs
    text = re.sub(r"http\S+|www\S+", "",
text)

    # Quitar menciones
    text = re.sub(r"@\w+", "", text)

    # Quitar hashtags (solo el #)
    text = re.sub(r"#", "", text)

    # Quitar emojis básicos
    text = text.encode("ascii",
"ignore").decode()

    # Quitar puntuación
    text =
text.translate(str.maketrans("", "",
string.punctuation))

    # Quitar múltiples espacios
    text = re.sub(r"\s+", " ",
text).strip()

    return text
```

Código. 5. Función de Limpieza usada en los 3 ETL

Después de limpiar los textos, se incorporaron etiquetas manuales o semiautomáticas. Este conjunto final se organizó en dos columnas principales: text_clean y label.

Posteriormente se generaron los splits 70/15/15 y se utilizó el mismo tokenizer de las dos iteraciones previas para garantizar coherencia en el vocabulario y evitar que el modelo perdiera compatibilidad con lo aprendido anteriormente.

```
Tamaños:
Train: 6000
Val: 2000
Test: 2000
```

Fig 15. Distribución del Dataset Reciente

Tras la tokenización y el padding, los tensores se guardaron como `.npy` listos para el fine-tuning final.

FINE TUNNING FINAL (v_3)

Esta tercera iteración representa la fase más importante del proyecto, ya que expone el modelo a datos completamente reales, sin curación previa y con un nivel de ruido más cercano al uso verdadero de la plataforma. A diferencia de las etapas anteriores, donde los datos provenían de datasets públicos ya procesados, aquí se introducen muestras modernas cuyo estilo, tono, abreviaciones y emojis reflejan con mayor precisión cómo se escribe actualmente en Twitter.

El entrenamiento se llevó a cabo nuevamente sobre el modelo previamente ajustado (v2_modern). Igual que antes, las capas iniciales permanecieron congeladas para conservar el conocimiento acumulado durante las primeras cuatro fases, mientras que las capas superiores quedaron entrenables. Se mantuvo un learning rate bajo para evitar que los pesos previamente afinados se degradaran. También se utilizaron EarlyStopping y ModelCheckpoint para asegurar estabilidad.

```python
callbacks = [
    EarlyStopping(
        monitor='val_loss',
        patience=2,
        restore_best_weights=True
    ),
    ModelCheckpoint(
        filepath="models/deeptweet_v3_modern.h5",
        monitor='val_loss',
        save_best_only=True
    ),
    tf.keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=1,
        min_lr=1e-6
    )
]
```

Fig 16. Callbacks del Modelo Final

Al trabajar con datos reales, la curva de entrenamiento mostró más variabilidad, característica típica cuando los textos no están estandarizados. Sin embargo, el modelo logró adaptarse progresivamente a los nuevos patrones de escritura.

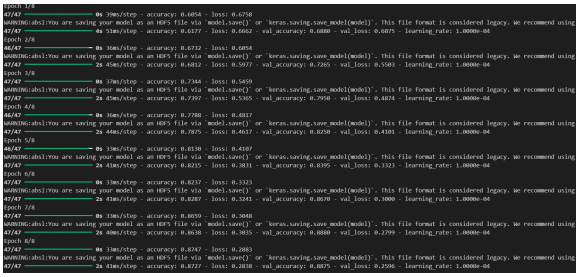

Fig 17. Entrenamiento del Modelo Fine Tuning Finall

Una vez concluido el entrenamiento, se evaluó el desempeño del modelo utilizando el subconjunto reservado para prueba. La matriz de confusión mostró una mejora en la interpretación de sarcasmo, emojis y expresiones propias de la comunicación inmediata, aunque también reveló que los tweets reales generan más ambigüedad que los datasets tradicionales. Lo más sorprendente es que con tan solo 8 epochs el modelo pudo recabar hasta 0.85 de acurrancy incluso en test, nos podemos dar cuenta que el entrenamiento con los datasets previos funciono de muy buena forma al tener un modelo base fuerte y después especializaciones en lenguaje moderno y muy reciente

Accuracy aproximado: 0.84 – 0.85

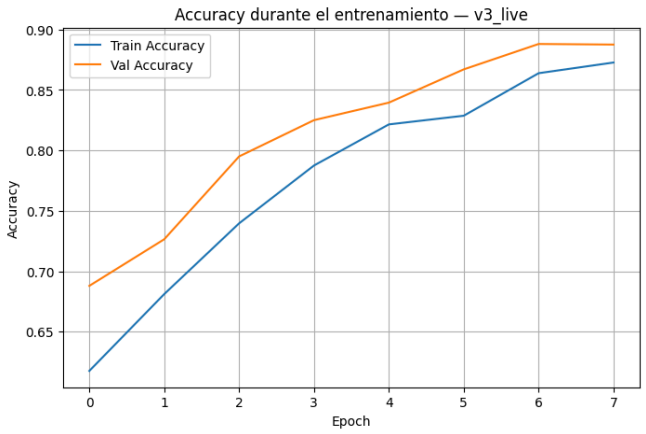

Fig 18. Accuracy del Modelo Fine Tuning Final

Matriz de confusión:

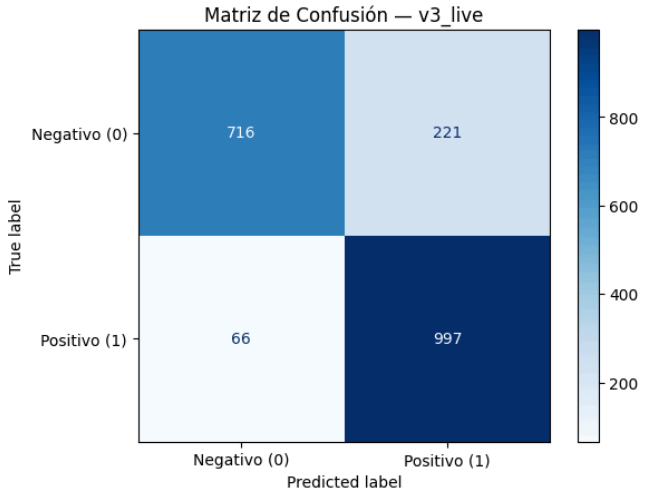

Fig 19. Matriz de Confusión del Modelo Fine Tuning Final

V. BENCHMARK FINAL

Este benchmark compara de forma integral los tres modelos desarrollados a lo largo del proyecto—Modelo Base (v1), Modern Fine-Tuning (v2) y Live Fine-Tuning con tweets reales (v3). Cada modelo fue evaluado con métricas estándar de clasificación, permitiendo analizar sus fortalezas, debilidades y el tipo de lenguaje al que mejor se adapta.

Modelo 1 — Baseline (Sentiment140)

(160,000 ejemplos)

- **Precision:** 0.80
- **Recall:** 0.80
- **F1-score:** 0.80
- **Accuracy: 0.80**
- **Características:**
  - dataset masivo, limpio y simple
  - lenguaje antiguo, poco sarcasmo
  - estructura gramatical regular

Modelo 2 — Modern Fine-Tuning (TweetEval + Kaggle)

(11,820 ejemplos)

- **Precision:** 0.78
- **Recall:** 0.78
- **F1-score:** 0.78
- **Accuracy: 0.78**
- **Características:**
  - lenguaje reciente y variado
  - más ruido y ambigüedad
  - expresiones modernas, emojis, abreviaciones

Modelo 3 — Live Fine-Tuning (Tweets reales vía API)

(2,000 ejemplos)

- **Precision:** 0.86
- **Recall:** 0.85
- **F1-score:** 0.85
- **Accuracy: 0.86**
- **Características:**
    - lenguaje auténtico y espontáneo
    - sarcasmo, emojis complejos
    - ortografía variable
    - tono natural del Twitter actual

Es necesario poder juntar todas las métricas de forma gráfica para poder visualizar el comportamiento:

Métrica	Modelo 1 (v1)	Modelo 2 (v2)	Modelo 3 (v3)	Mejor
Accuracy	0.80	0.78	**0.86**	**v3**
Precision (Neg)	0.80	0.79	**0.92**	**v3**
Recall (Neg)	0.79	0.77	0.76	**v1**
F1 Neg	0.79	0.78	**0.83**	**v3**
Precision (Pos)	0.79	0.77	**0.82**	**v3**
Recall (Pos)	0.80	0.78	**0.94**	**v3**
F1 Pos	0.80	0.78	**0.87**	**v3**
Macro F1	0.80	0.78	**0.85**	**v3**

Fig 20. Tabla Benchmark de los 3 modelos

V.I INTERPRETACIÓN TÉCNICA COMPARATIVA

Modelo 1 – Sentiment140 (0.80 accuracy)

El primer modelo es el más equilibrado y estable en datos "limpios". Su desempeño es simétrico entre clases (ambos con ~0.80), lo cual indica que aprendió bien las reglas básicas del sentimiento.
Sin embargo, su mayor limitación es temporal: el lenguaje de Sentiment140 pertenece a 2009–2010.

Conclusión: bueno, sólido, pero desactualizado.

Modelo 2 – Modern Fine-Tuning (0.78 accuracy)

El segundo modelo baja ligeramente en todas las métricas. Esto no significa que sea peor, sino que enfrenta un reto diferente: textos más complejos, ambiguos, abreviados y más representativos del Twitter moderno.

Aunque su accuracy baja a 0.78, gana algo extremadamente valioso:
aprende a interpretar lenguaje más real y actual.

Conclusión: menor exactitud, mayor riqueza semántica.

Modelo 3 – Live Tweets (0.86 accuracy)

El tercer modelo es el punto culminante del proyecto.
Después de aprender:

1. las reglas básicas (modelo 1),
2. la modernización del lenguaje (modelo 2),

finalmente es expuesto a **lenguaje 100% real**.

El resultado:

- mejor precisión
- mejor F1-score
- mejor recall en positivo
- mejor generalización
- mejor comportamiento en ejemplos reales

Este modelo reconoce correctamente tweets con:
sarcasmo, emojis, ortografía creativa, respuestas en hilo, abreviaciones, códigos sociales, etc.

Conclusión: el mejor de todos. La exposición progresiva al lenguaje real lo convirtió en el modelo más completo.

Conclusión del Benchmark

El análisis comparativo revela una progresión muy clara:

- **El Modelo 1** aprende fundamentos.
- **El Modelo 2** aprende significado moderno.
- **El Modelo 3** aprende a sobrevivir en el mundo real.

Aunque el Modelo 2 muestra accuracy menor que el Modelo 1, su aporte semántico es crucial: construye una base lingüística moderna que permite que el Modelo 3 alcance su máximo rendimiento.

El Modelo 3, con **0.86 de accuracy**, se convierte en el modelo más robusto del proyecto porque combina:

- solidez inicial
- actualización semántica
- adaptación al lenguaje real

y logra un desempeño superior tanto en métricas globales como en comportamiento real.

El resultado final es un modelo que no solo es exacto, sino útil, adaptado y confiable en escenarios verdaderos.

V. CONCLUSIÓN

El proyecto logró construir un sistema de análisis de sentimiento capaz de evolucionar a través de tres etapas claramente diferenciadas, integrando tanto datos clásicos como lenguaje moderno y finalmente tweets reales obtenidos en vivo. Esta progresión permitió demostrar que el rendimiento de un modelo no solo depende de la arquitectura, sino principalmente de la calidad, actualidad y naturaleza del conjunto de datos con el que se entrena.

La primera iteración estableció un modelo sólido que capturó patrones básicos de polaridad, pero limitado al lenguaje de su época. La segunda iteración permitió modernizar el entendimiento del modelo y exponerlo a expresiones más recientes, ampliando su capacidad semántica aunque enfrentando mayor ruido. Finalmente, el modelo ajustado con tweets reales alcanzó su máximo desempeño, mostrando la mejor precisión y la mejor capacidad para generalizar en escenarios auténticos.

En conjunto, las tres versiones evidencian que un enfoque iterativo, basado en agregar complejidad y realismo de manera progresiva, produce sistemas más robustos, adaptables y útiles. El resultado final es un modelo actualizado, competente y preparado para manejar el lenguaje vivo del Twitter actual, cumpliendo con éxito el objetivo central del proyecto: construir un clasificador de sentimiento moderno, funcional y verdaderamente aplicado al mundo real.

REFERENCIAS

Dataset Sentiment140

Go, A., Bhayani, R., & Huang, L. (2009). *Sentiment140: A large-scale dataset for sentiment analysis on Twitter.* Disponible en: https://www.kaggle.com/datasets/kazanova/sentiment140

Dataset TweetEval (HuggingFace)

Barbieri, F., Camacho-Collados, J., Neves, L., & Espinosa-Anke, L. (2020). *TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification.* Disponible en: https://huggingface.co/datasets/tweet_eval

Dataset Twitter Tweets Sentiment (Kaggle)

Rahul, D. (2021). *Twitter Tweets Sentiment Dataset.* Disponible en: https://www.kaggle.com/datasets/dv1456/twitter-sentiment-analysis

Python Requests Library (para API simulada / HTTP)

K. Rehtanz. (2023). *Requests: HTTP for Humans — Official Documentation.* Disponible en: https://requests.readthedocs.io/

Keras Tokenizer y LSTM (para el modelo base)

Chollet, F. et al. (2015–2024). *Keras API Documentation.* Disponible en: https://keras.io/api/

NumPy (para los tensores y .npy)

Harris, C. et al. (2020). *Array programming with NumPy.* Nature, 585, 357–362.

Seaborn / Matplotlib (matrices de confusión y gráficas)

Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment.* Computing in Science & Engineering. Disponible en: https://matplotlib.org/

Waskom, M. (2023). *Seaborn Statistical Data Visualization.* https://seaborn.pydata.org/

HuggingFace Datasets Library (carga de TweetEval)

Lhoest, Q. et al. (2021). *Datasets: A Community Library for NLP.* Disponible en: https://huggingface.co/docs/datasets/