

# Proyecto Semestral: Clasificador de Frutas en Imágenes y Clasificador de Noticias

Claudio Cárcamo, Jairo Millapán, Maximiliano Romero, Daniela Salinas

COM4402 – Introducción a Inteligencia Artificial

Escuela de Ingeniería, Universidad de O'Higgins

22, Diciembre, 2023

**Abstract**— Durante este proyecto abordaremos los conceptos y realizaremos un análisis de aplicaciones actuales de Inteligencia Artificial, basado en métodos de Deep Learning. Específicamente en Redes Neuronales Convolucionales(CNN) y Redes Neuronales Recurrentes(RNN) a través de la clasificación de frutas y noticias. Reportamos los resultados más relevantes y los interpretamos con el fin de discutir sobre la calidad de ellos.

**Keywords**— Redes Neuronales Convolucionales, Redes Neuronales Recurrentes, Clasificación, Deep Learning, Inteligencia Artificial.

## I. INTRODUCTION

Hoy en día, podemos notar que la Inteligencia Artificial está presente en distintas áreas como industrial, robótica, aprendizaje automático, visión por computadora, etc. Nos enfocaremos en lo que es la visión por computadora, donde, usaremos estas herramientas para la clasificación de frutas en imágenes a través de la inspección automatizada. Se destaca la extracción de características de imágenes, como textura y color, proponiendo un clasificador(véase Fig. 1 y 2)..



Fig. 1 Diferentes colores y formas de frutas



Fig. 2 Ejemplo clasificación de la inteligencia artificial

Por otro lado, el clasificador de noticias es una herramienta que utiliza algoritmos y modelos de aprendizaje automático para organizar y etiquetar automáticamente noticias en categorías específicas. Con el objetivo de analizar información de manera eficiente y simplificar la recuperación y búsqueda de noticias.

En esta ocasión aplicaremos estos clasificadores usando las Redes Neuronales Convolucionales y Recurrentes(CNN y RNN), donde las Convolucionales es una arquitectura de red para Deep Learning que aprende directamente a partir de datos. Son particularmente útiles para identificar patrones en imágenes con el fin de reconocer objetos, clases y categorías. Mientras que las Recurrentes es un tipo de red neuronal artificial que utiliza datos secuenciales o datos de series temporales(véase Fig. 3 y 4).

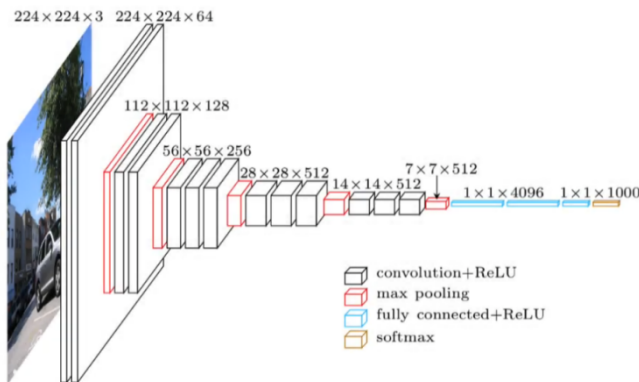


Fig. 3 Funcionamiento de una red neuronal convolucional

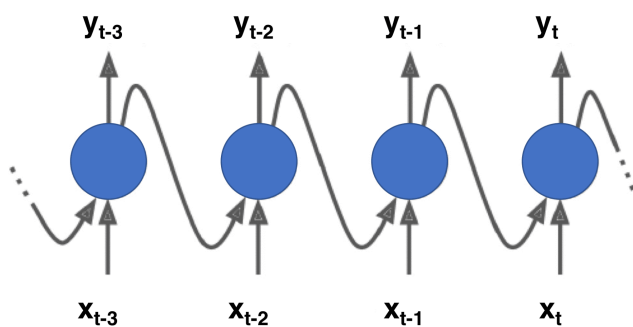


Fig. 4 Funcionamiento de una red neuronal recurrente

Ahora, para poder implementar los clasificadores usaremos el lenguaje de programación Python que nos ayudará a implementar estas redes neuronales convolucionales a código mediante la librería “TensorFlow” que es una de las librerías que facilita la creación de modelos de aprendizaje automático para nuestros computadores, permitiéndonos en nuestro proyecto construir nuestras redes convolucionales y recurrentes para detectar patrones y crear los clasificadores correspondientes.

## II. THEORETICAL FRAMEWORK

El aprendizaje profundo se centra en los algoritmos y arquitecturas de redes neuronales, brindándonos técnicas y modelos más avanzados de inteligencia artificial que nos permiten trabajar con una gran cantidad de datos. En esta ocasión, resolveremos varios problemas basándonos en los fundamentos de la inteligencia artificial, el Deep learning y la aplicación de arquitecturas de CNN y RNN interpretadas por Python.

### A. Deep learning.

El aprendizaje profundo es un subconjunto del Machine Learning que busca emular el comportamiento del cerebro reflejado en las tecnologías. Necesitamos máquinas que sean capaces de auto programarse, queremos máquinas que aprendan de su propia experiencia, estas están basadas y programadas en algoritmos que pueden aprender mediante grandes bases de datos sin intervención humana previa, sacando ellos mismos las conclusiones según la información proporcionada.

### B. Preparación de los datos.

Cuando hablamos de preparación de los datos en la clasificación de fruta nos referimos a el proceso en donde carga todas las imagen, luego las separa en distintos conjuntos, uno de entrenamiento y uno de prueba, se separan las características y se etiqueta cada conjunto de datos, luego se ajustan las dimensiones y se normalizan los datos, similar al caso del clasificador de noticia en donde en una variable guardamos el nombre de las clases, en otra los numero y en otra el número total de ejemplos de cada conjunto, todo esto nos permitirá analizar e interpretar los datos para luego evaluar el rendimiento en los modelos de redes neuronales para estudiar datos en diferentes categorías o clases y los gráficos son representaciones visuales del rendimiento de un modelo de red neuronal.

### C. Arquitectura de CNN.

Las CNN(Convolutional Neuronal Network) también conocidas como redes neuronales convolucionales, son modelos que permiten principalmente el reconocimiento de imágenes asignando una etiqueta correspondiente a la clase a la que pertenece cada imagen proporcionada en la entrada. Su estructura se compone de dos partes: una parte convolucional que comprime las características propias de cada imagen para reducir su tamaño inicial; y una parte de clasificación, que envía el código CNN de la salida de la parte convolucional a una segunda parte, que está compuesta por capas totalmente conectadas conocidas como perceptrón multicapa.

### D. Arquitectura de RNN.

Las Redes Neuronales Recurrentes son una de las principales arquitecturas basadas en el Machine

Learning, en su mayoría es utilizada en sistemas de reconocimiento de voz o en el análisis video, o en el procesamiento del lenguaje natural. Usa datos secuenciales o de series de tiempo. Los problemas ordinales o temporales, como la traducción de idiomas, y los subtítulos de imágenes, se abordan con frecuencia con estos algoritmos de aprendizaje profundo, a diferencia de otras arquitecturas que pueden analizar solo una cosa a la vez esta puede interpretar secuencias de sonidos o imágenes, estas no se basan solo en clasificar un dato en particular, sino que también forman otras secuencias

### III. METHODOLOGY

En esta ocasión hablaremos sobre los materiales y métodos para llevar a cabo esta investigación.

### A. Materials

- **Google Colab**, es un medio que nos entrega Google para programar en distintos lenguajes, aquí estará guardado todo el código que usamos para realizar el análisis y obtener los resultados(véase Fig. 5).
- **Python**, lenguaje de programación que implementaremos en Google Colab.
- **Bases de datos**, como ya mencionamos, necesitamos las bases de datos a la cual le realizaremos dicho análisis.
- **Pandas**, librería de Python especializada en la manipulación y el análisis de datos(véase Fig. 6).
- **Numpy**, librería de Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas(véase Fig. 6).
- **Keras**, librería de Python diseñada específicamente para hacer experimentos con redes neuronales(véase Fig. 6).
- **TensorFlow**, librería de Python que te ayuda a gestionar e implementar los procesos de aprendizaje automático(véase Fig. 6).

- **Matplotlib.pyplot**, librería de Python que permite visualizar y graficar datos (véase Fig. 6).



Fig. 5 Símbolo de Google Colab

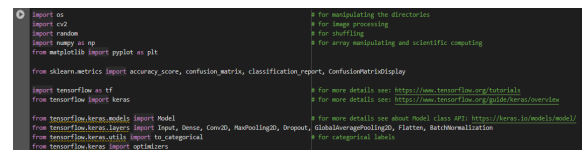


Fig. 6 Librerías importadas en Google Colab

### B. Métodos Clasificador de Frutas

Primero que todo, debemos ejecutar las celdas donde importamos las librerías, definimos nuestras variables generales y donde descargamos el conjunto de datos de imágenes de frutas. Después, procedemos a ejecutar nuestras celdas que están encargadas de leer el conjunto de entrenamiento. En ellas, establecemos el directorio de entrenamiento de la ruta gracias al comando *os\_path\_join*, creamos un ciclo para iterar sobre cada categoría y otro para iterar sobre cada imagen de cada categoría con el fin de leer las imágenes e ir guardándolas en la variable *train\_images* la cual hace referencia donde guardamos las imágenes del conjunto de entrenamiento (véase Fig. 7). Además, en la siguiente celda implementaremos código complementado con la librería Matplotlib con el fin de graficar aleatoriamente algunas de nuestras imágenes recolectadas (véase Fig. 8).

```
[19] # Read training set
train_images = []
train_dir = os.path.join(base_dir, 'Training/')

# set the training directory in the path

for category in CATEGORIES:
    # iterate to each category
    path = os.path.join(train_dir, category)
    class_num = CATEGORIES.index(category)
    for image in os.listdir(path):
        # iterate to each image in the category
        if image.endswith('.jpg') and not image.startswith('.'):
            # read the image
            img_array = cv2.imread(os.path.join(path, image),
                                   cv2.IMREAD_GRAYSCALE)
            # save the image in training data array
            train_images.append([img_array, class_num])

print('Training images: ', len(train_images))
```

Fig. 7 Código utilizado para leer el conjunto de entrenamiento.

```
import random
img_idx = random.sample(range(len(train_images)), 25)
img_idx = np.array(img_idx)

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[img_idx[i]][0], cmap='gray')
    plt.xlabel(class_names[train_images[img_idx[i]][1]])
plt.show()
```

Fig. 8 Código utilizado para graficar aleatoriamente las imágenes del conjunto de entrenamiento.

A continuación debemos repetir los códigos de las figuras 7 y 8 con la diferencia de que lo haremos con el conjunto de datos de prueba (véase Fig 9 y 10).

```
[21] # Read testing set
test_images = []
test_dir = os.path.join(base_dir, 'Test/')

for category in CATEGORIES:
    path = os.path.join(test_dir, category)
    class_num = CATEGORIES.index(category)
    for image in os.listdir(path):
        if image.endswith('.jpg') and not image.startswith('.'):
            img_array = cv2.imread(os.path.join(path, image),
                                   cv2.IMREAD_GRAYSCALE)
            test_images.append([img_array, class_num])

print("Testing images: ", len(test_images))
```

Fig. 9 Código utilizado para leer el conjunto de prueba

```
import random
img_idx = random.sample(range(len(test_images)), 25)
img_idx = np.array(img_idx)

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[img_idx[i]][0], cmap='gray')
    plt.xlabel(class_names[test_images[img_idx[i]][1]])
plt.show()
```

Fig. 10 Código utilizado para graficar aleatoriamente las imágenes del conjunto de prueba

Antes de empezar el entrenamiento de los datos, aplicaremos un poco de código para mezclar los datos y obtener una mejor precisión. Lo realizaremos a través del comando *random.shuffle* que utilizaremos en cada conjunto y usaremos un ciclo para dividir las

características y etiquetas de cada conjunto (véase Fig. 11).

```
# Shuffle the dataset before training for better accuracy
x_train = []
y_train = []

random.shuffle(train_images)

for features, label in train_images:
    x_train.append(features)
    y_train.append(label)
x_train = np.array(x_train)

x_test = []
y_test = []

random.shuffle(test_images)

for features, label in test_images:
    x_test.append(features)
    y_test.append(label)
x_test = np.array(x_test)
```

Fig. 11 Código utilizado para mezclar datos

El último paso antes de efectuar el entrenamiento es remodelar y normalizar los datos antes del entrenamiento. Los comandos que participan en esta oportunidad son *reshape* y *np.mean* (véase Fig. 12).

```
# reshape and normalize the data before training
x_train = x_train.reshape(-1, img_size, img_size, 1)
mean_train = np.mean(x_train, axis=0)
x_train = x_train - mean_train
x_train = x_train / 255

x_test = x_test.reshape(-1, img_size, img_size, 1)
mean_test = np.mean(x_test, axis=0)
x_test = x_test - mean_test
x_test = x_test / 255

y_train = np.asarray(y_train)
y_test = np.asarray(y_test)

print(x_train.shape)
print(x_test.shape)
```

Fig. 12 Código utilizado para remodelar y normalizar datos.

Ahora con todas esas configuraciones, empezaremos a definir nuestros parámetros para crear nuestro modelo y luego entrenarlo. Definiremos la forma de entrada de las imágenes, los números de filtros para cada capa, el tamaño de los filtros, el tamaño de la ventana para las operaciones de pooling, el término de regularización L2 para controlar el sobreajuste, la tasa de abandono para las capas, la tasa de aprendizaje para el optimizador, el valor de momentum para el optimizador, el número de épocas de entrenamiento, el tamaño del lote y creamos un objeto de regularización L2 utilizando el valor



regularización L2 para controlar el sobreajuste definido anteriormente mediante el método *keras.regularizers.l2* (véase Fig. 13).

```
[25] # Hyperparameters settings
input_shape = x_train.shape[1:]
filters_numbers = [16, 32, 64]
filters_size = [[5,5],[4,4],[3,3]]

pool_size=(2, 2)
weight_decay = 5e-4
dropout = 0.6
lr = 0.001
momentum = 0.9

epochs = 10
batch_size = 32

L2_norm = keras.regularizers.l2(weight_decay)
```

Fig. 13 Código utilizado para configurar los hiperparámetros

Ya podemos crear y configurar nuestro modelo de red neuronal convolucional (CNN), donde definiremos lo siguiente: una capa de entrada, 3 capas convolucionales, maxpooling, pooling global promedio en todas las dimensiones espaciales, flatten, capa completamente conectada con 512 unidades y activación lineal, capa de dropout que apaga aleatoriamente un porcentaje de unidades para prevenir el sobreajuste y capa de salida con el número de unidades igual al número de clases en el problema y función de activación softmax. Creamos el modelo y mostramos un resumen del modelo creado (véase Fig. 14).

```
# Setup model layers
# Input layer
model_input = Input(shape=input_shape)

# 1st Convolutional layer
model_output = Conv2D(filters_numbers[0], kernel_size=filters_size[0], kernel_regularizer=L2_norm, padding='same',
activation='relu', data_format='channels_last')(model_input)
model_output = BatchNormalization()(model_output)
model_output = MaxPooling2D(pool_size=pool_size)(model_output)

# 2nd Convolutional layer
model_output = Conv2D(filters_numbers[1], kernel_size=filters_size[1], kernel_regularizer=L2_norm, padding='same',
activation='relu', data_format='channels_last')(model_output)
model_output = BatchNormalization()(model_output)
model_output = MaxPooling2D(pool_size=pool_size)(model_output)

# 3rd Convolutional layer
model_output = Conv2D(filters_numbers[2], kernel_size=filters_size[2], kernel_regularizer=L2_norm, padding='same',
activation='relu', data_format='channels_last')(model_output)
model_output = BatchNormalization()(model_output)
model_output = GlobalAveragePooling2D(data_format='channels_last')(model_output)

# Convert features to Flatten vector
model_output = Flatten()(model_output)

# Full-connected layer
model_output = Dense(512)(model_output)
model_output = Dropout(dropout)(model_output)

# Output layer
model_output = Dense(num_classes, activation='softmax', name='id')(model_output)

# Create the Model by using Input and Output layers
model = Model(inputs=model_input, outputs=model_output, name=NAME)

# Show the Model summary information
model.summary()

# Compile the model
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), optimizer=optimizers.SGD(lr, momentum), metrics=['accuracy'])
```

Fig. 14 Código utilizado para crear modelo de red neuronal convolucional

Finalmente podemos entrenar nuestro modelo con el comando *fit* (véase Fig. 15).

```
[27] # Train the model
print("[INFO] Train the model on training data")

history = model.fit(x=x_train, y=np.asarray(y_train), batch_size=batch_size, epochs=epochs, verbose=1, validation_split=0.1)
```

Fig. 15 Código utilizado para entrenar el modelo

Para terminar graficamos las curvas de entrenamiento y validación del modelo después de entrenar. También, testeamos nuestro modelo a través de los comandos *evaluate* y *predict*. Terminamos con una matriz de confusión para evaluar el rendimiento del modelo en el conjunto de prueba (véase Fig. 16).

```
# Plot training curves
x = [1,2,3,4,5,6,7,8,9,10]
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.xticks(x)
plt.legend(['Train', 'Validation'], loc='lower right')
plt.grid(b=None, which='major', axis='both')
plt.savefig('fruits_classifier-training_acc.png')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.xticks(x)
plt.legend(['Train', 'Validation'], loc='upper right')
plt.grid(b=None, which='major', axis='both')
plt.savefig('fruits_classifier-training_loss.png')
plt.show()

# Test the model
print("[INFO] Evaluate the test data")

results = model.evaluate(x=x_test, y=y_test, batch_size=batch_size, verbose=1)
print('Testing Loss, Testing Acc: ', [round(r,4) for r in results])

y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)

print('Accuracy', round(accuracy_score(y_test, y_pred),4))
print('Classification report', classification_report(y_test, y_pred, target_names=class_names))

# Plot Testing confusion matrix
cmat = confusion_matrix(np.asarray(y_test), y_pred, normalize='true')
disp = ConfusionMatrixDisplay(confusion_matrix=cmat, display_labels=class_names)
disp.plot(cmap='Blues', xticks_rotation='vertical', values_format='.2f')
plt.show()
```

Fig. 16 Código utilizado para graficar, evaluar y generar una matriz de confusión del modelo

### C. Métodos Clasificador de Noticias

Nuevamente, lo primero que debemos hacer es cargar las librerías que nos ayudarán durante este clasificador de noticias con red neuronal recurrente. Inicialmente, el conjunto de entrenamiento contiene 120.000 muestras de noticias, mientras que el conjunto de prueba contiene 7600 muestras. Tomaremos un 10% de las noticias del

conjunto de entrenamiento y las pasaremos al conjunto de validación para aumentarlo un poco (véase Fig. 17).

```
[ ] (train_data, val_data), info = tfds.load("ag_news_subset:1.0.0", #version 1.0.0
    split=["train[:90%]", "train[90%:] + test"],
    with_info=True,
    as_supervised=True
)
```

Fig. 17 Código utilizado para cargar los datos y dividirlos según la proporción indicada

Seguiremos con crear una variable llamada *class\_names* que guardará los nombres de las clases y la variable *num\_classes* que guardará el número de las clases mediante el método *features* y después guardaremos el número total de ejemplos del conjunto de entrenamiento y de validación en las variables *num\_train* y *num\_val* respectivamente con el método *splits*. También vamos a visualizar las 10 primeras muestras de noticias utilizando el método *tfds.as\_dataframe* para mostrarlas como dataframe (véase Fig. 18).

```
[ ] # Displaying the classes
class_names = info.features['label'].names
num_classes = info.features['label'].num_classes
print(f'The news are grouped into {num_classes} classes that are :{class_names}')

[ ] num_train = info.splits['train'].num_examples
num_val = info.splits['test'].num_examples
print(f'The number of training samples: {num_train} \n The number of validation samples: {num_val}')

También podemos visualizar las 10 primeras muestras de noticias. Podemos utilizar tfds.as_dataframe para mostrarlas como dataframe.

[ ] news_df = tfds.as_dataframe(train_data.take(10), info)
news_df.head(10)

Vamos a mostrar algunas noticias completas.

[ ] for i in range(0,4):
    print(f'Sample news {i}\n \
    label: {news_df['label'][i]} ({class_names[i]})\n \
    Description: {news_df['description'][i]}\n-----\n')
```

Fig. 18 Código utilizado para guardar variables y visualizar noticias

Prepararemos nuestros conjuntos de datos donde el primer paso es mezclar y agrupar los datos de entrenamiento (véase Fig. 19).

```
[8] buffer_size = 1000
    batch_size = 32

    train_data = train_data.shuffle(buffer_size)
    train_data = train_data.batch(batch_size).prefetch(1)
    val_data = val_data.batch(batch_size).prefetch(1)

[9] for news, label in train_data.take(1):
    print(f'Sample news\n----\n {news.numpy()[0:4]} \n----\nCorresponding labels: {label.numpy()[0:4]}')
```

Fig. 19 Código utilizado para mezclar y agrupar los datos de entrenamiento

Ahora, podemos utilizar Keras TextVectorization layer para manejar todo el preprocesamiento de texto

necesario. Convertirá los textos en tokens, los convertirá en secuencias, rellenará las secuencias. También elimina puntuaciones y minúsculas. Después de crear la capa, podemos utilizar adapt para pasar el conjunto de datos a través de ella. Fíjate en que usamos lambda function para obtener la descripción separada de la etiqueta. Podemos obtener el vocabulario. El vocabulario es la lista de palabras individuales que componen una frase concreta (véase Fig. 20).

```
[10] max_features = 10000
    text_vectorizer = tf.keras.layers.TextVectorization(max_tokens=max_features)

[11] text_vectorizer.adapt(train_data.map(lambda description, label : description))

[12] vocab = text_vectorizer.get_vocabulary()
    vocab[:10]

['', '[UNK]', 'the', 'a', 'to', 'of', 'in', 'and', 'on', 'for']

[13] sample_news = ['This weekend there is a sport match between Man U and Fc Barcelona',
    'Tesla has unveiled its humanoid robot that appeared dancing during the show!']

[14] vectorized_news = text_vectorizer(sample_news)
    vectorized_news.numpy()

array([[ 40, 494, 186, 16, 3, 1567, 570, 159, 370, 1, 7,
    7486, 2556],
    [ 1, 20, 876, 13, 1, 4845, 10, 1273, 1, 160, 2,
    532, 0]])
```

Fig. 20 Código utilizado para preparar los datos

Procedemos a crear nuestro modelo de red neuronal recurrente el cual consistirá de: capa de vectorización de texto, capa de incrustación que mapea números enteros, dos capas de convolución unidimensional, capa de pooling unidimensional, capa de pooling global máximo unidimensional, capa totalmente conectada con 32 unidades y activación ReLU, capa de dropout y su capa de salida (véase Fig. 21).


```
[16] model = tf.keras.Sequential([
    text_vectorizer,
    tf.keras.layers.Embedding(input_dim=input_dim, output_dim=64, mask_zero=True),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.GlobalMaxPool1D(),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(4, activation='softmax')
])
```

Fig. 21 Código utilizado para crear el modelo de red neuronal recurrente


Mostraremos un resumen de nuestro modelo creado y mostraremos la forma de trazarlo con el método *Keras util's plot\_model* (véase Fig. 22).

```

Obtener el resumen del modelo

✓  model.summary()

También podemos trazar el modelo con Keras util's plot_model.

✓  [18] from tensorflow.keras.utils import plot_model

plot_model(model)

```

Fig. 22 Código utilizado para el resumen y trazado del modelo

Continuaremos con el entrenamiento de nuestro modelo, donde primero debemos compilarlo, definir ciertos parámetros y aplicarle el método fit(véase Fig. 23).

```

[19] # Compile the model

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

[20] batch_size = 32

train_steps = int(len(train_data)/batch_size)
val_steps = int(len(val_data)/batch_size)

[21] # Train the model

history = model.fit(train_data,
                    epochs=25,
                    validation_data=val_data,
                    steps_per_epoch=train_steps,
                    validation_steps=val_steps
                    )

```

Fig. 23 Código utilizado para el entrenamiento del modelo

Para analizar los errores trazaremos las curvas de la pérdida y la precisión en el transcurso de las épocas gracias a la librería Matplotlib(véase Fig. 24).

```

[22] import matplotlib.pyplot as plt

# function to plot accuracy and loss

def plot_acc_loss(history):

    model_history = history.history
    acc = model_history['accuracy']
    val_acc = model_history['val_accuracy']
    loss = model_history['loss']
    val_loss = model_history['val_loss']
    epochs = history.epoch

    plt.figure(figsize=(10,5))
    plt.plot(epochs, acc, 'r', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
    plt.title('Training and validation accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc=0)

# Create a new figure with plt.figure()
plt.figure()

plt.figure(figsize=(10,5))
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'y', label='Validation Loss')
plt.title('Training and validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc=0)
plt.show()

```

Fig. 24 Código utilizado para graficar las curvas de pérdida y precisión

La idea es combinar CNN y RNN para clasificar los textos con el fin de que RNN pueda dotar a la red de la capacidad de manejar secuencias. Así, Conv1D puede extraer palabras significativas en las frases de entrada, y las RNN pueden prever la secuencia contenida en las características de salida de las CNN. Para esto, creamos nuestro modelo idéntico al de la figura 21 pero agregando una capa LSTM bidireccional para manejar las características de las CNNs desde ambas direcciones(véase Fig. 25).

```

conv_rnn_model = tf.keras.Sequential([
    TextVectorizer,
    tf.keras.layers.Embedding(input_dim=input_dim, output_dim=64, mask_zero=True),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.MaxPooling1D(),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(4, activation='softmax')
])

```

Fig. 25 Código utilizado para crear el modelo con la capa adicional

Seguimos con los pasos, mostrar un resumen del modelo, compilarlo, definir parámetros y entrenarlo(véase Fig. 26). Además, visualizamos los gráficos con el método `plot_acc_loss(history)`.

```
[ ] conv_rnn_model.summary()

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

[ ] batch_size = 32
train_steps = int(len(train_data)/batch_size)
val_steps = int(len(val_data)/batch_size)

# Train the model
history = conv_rnn_model.fit(train_data,
                             epochs=25,
                             validation_data=val_data,
                             steps_per_epoch=train_steps,
                             validation_steps=val_steps
                             )
```

Fig. 26 Código utilizado para el entrenamiento del modelo

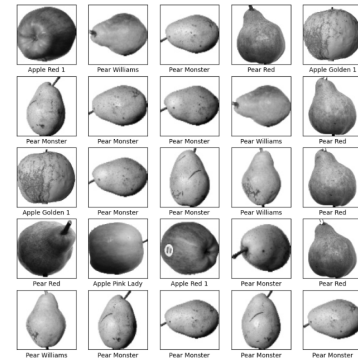


Fig.29 Resultados al ejecutar celdas de las figuras 9 y 10

Para probar nuestra red, crearemos la función predict, donde su finalidad es realizar predicciones utilizando el modelo sobre la muestra de noticias y mostrar la clase predicha junto con su nombre correspondiente. Ya creada nuestra función, la llamamos con distintas noticias para probarla(véase Fig. 27).

```
[ ] def predict(model, sample_news, class_names):
    # Convert sample news into array
    sample_news = np.array(sample_news)
    # Predict the news type
    preds = model.predict(sample_news)
    pred_class = np.argmax(preds)
    print('predicted class: (pred_class) Unpredicted class name: (class_names[pred_class])')

    sample_news = ["Tesla, a self driving car company is also planning to make a humanoid robot. This humanoid robot appeared dancing in the latest Tesla AI day"]
    predict(conv_rnn_model, sample_news, class_names)

    sample_news = ["To the last week, there has been many transfer surprises in football. Ronaldo went back to old Trafford, "
    "while messi went to Paris Saint Germain to join his former colleague Neymar."
    "We can't wait to see these two clubs will perform in upcoming leagues"]

    predict(conv_rnn_model, sample_news, class_names)

    sample_news = ["In the latest business news: the tech giant NVIDIA has acquired ARM, a microprocessor company"]
    predict(conv_rnn_model, sample_news, class_names)
```

Fig. 27 Código utilizado para probar nuestro modelo

#### IV. RESULTS

Al ejecutar las celdas mostradas en las Fig. 7, 8, 9 y 10. Tendremos como resultado las imágenes elegidas aleatoriamente de cada conjunto(véase Fig. 28 y 29).

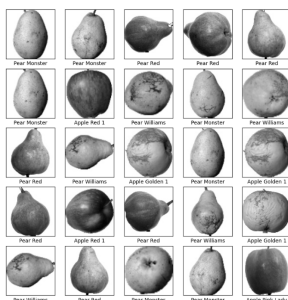


Fig. 28 Resultados al ejecutar celdas de las figuras 7 y 8

Model: "fruits-classifier"		
Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 100, 100, 1)]	0
conv2d_6 (Conv2D)	(None, 100, 100, 16)	416
batch_normalization_6 (Batch Normalization)	(None, 100, 100, 16)	64
max_pooling2d_4 (MaxPooling2D)	(None, 50, 50, 16)	0
conv2d_7 (Conv2D)	(None, 50, 50, 32)	8224
batch_normalization_7 (Batch Normalization)	(None, 50, 50, 32)	128
max_pooling2d_5 (MaxPooling2D)	(None, 25, 25, 32)	0
conv2d_8 (Conv2D)	(None, 25, 25, 64)	18496
batch_normalization_8 (Batch Normalization)	(None, 25, 25, 64)	256
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 64)	0
flatten_2 (Flatten)	(None, 64)	0
dense_2 (Dense)	(None, 512)	33280
dropout_2 (Dropout)	(None, 512)	0
id (Dense)	(None, 6)	3078
Total params: 63942 (249.77 KB)		
Trainable params: 63718 (248.90 KB)		
Non-trainable params: 224 (896.00 Byte)		

Fig. 30 Resumen del modelo creado de red neuronal convolucional

Después, al ejecutar la celda de la figura 15, nos mostrará los resultados del entrenamiento del modelo(véase Fig 31).

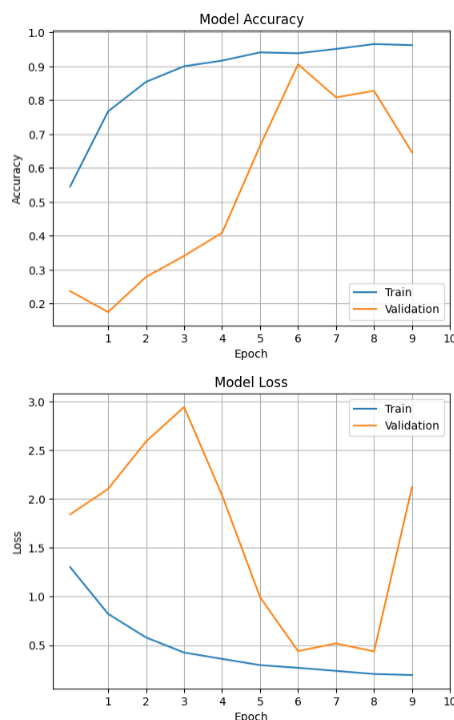


```

[INFO] Train the model on training data
Epoch 1/10
87/87 [=====] - 31s 334ms/step - loss: 1.3013 - accuracy: 0.5456 - val_loss: 1.0429 - val_accuracy: 0.2370
Epoch 2/10
87/87 [=====] - 26s 298ms/step - loss: 0.8219 - accuracy: 0.7672 - val_loss: 2.1040 - val_accuracy: 0.1753
Epoch 3/10
87/87 [=====] - 27s 310ms/step - loss: 0.5793 - accuracy: 0.8543 - val_loss: 2.5916 - val_accuracy: 0.2792
Epoch 4/10
87/87 [=====] - 27s 305ms/step - loss: 0.4248 - accuracy: 0.9002 - val_loss: 2.9445 - val_accuracy: 0.3409
Epoch 5/10
87/87 [=====] - 27s 309ms/step - loss: 0.3598 - accuracy: 0.9168 - val_loss: 2.8391 - val_accuracy: 0.4091
Epoch 6/10
87/87 [=====] - 27s 305ms/step - loss: 0.2958 - accuracy: 0.9411 - val_loss: 0.9930 - val_accuracy: 0.6656
Epoch 7/10
87/87 [=====] - 33s 373ms/step - loss: 0.2683 - accuracy: 0.9385 - val_loss: 0.4392 - val_accuracy: 0.9058
Epoch 8/10
87/87 [=====] - 26s 304ms/step - loss: 0.2366 - accuracy: 0.9512 - val_loss: 0.5188 - val_accuracy: 0.8884
Epoch 9/10
87/87 [=====] - 27s 305ms/step - loss: 0.2048 - accuracy: 0.9657 - val_loss: 0.4360 - val_accuracy: 0.8279
Epoch 10/10
87/87 [=====] - 26s 297ms/step - loss: 0.1941 - accuracy: 0.9624 - val_loss: 2.1238 - val_accuracy: 0.6461
    
```

Fig. 31 Resultados entrenamiento del modelo de red neuronal convolucional

Los últimos resultados que veremos en este código de clasificador de frutas en imágenes serán al ejecutar las celdas de la figura 16, donde veremos las curvas de entrenamiento y validación del modelo después de entrenar, la evaluación del modelo y su matriz de confusión (véase Fig. 32).



```

[INFO] Evaluate the test data
33/33 [=====] - 3s 58ms/step - loss: 1.5554 - accuracy: 0.6602
Testing Loss, Testing Acc: [1.5554, 0.6602]
33/33 [=====] - 3s 72ms/step
Accuracy 0.6602
Classification report
precision    recall  f1-score   support

Apple Golden 1      0.90      0.59      0.71      160
Apple Pink Lady      0.00      0.00      0.00      152
Apple Red 1         1.00      0.77      0.87      164
Pear Red            0.47      1.00      0.64      222
Pear Williams       0.78      0.66      0.71      166
Pear Monster        0.70      0.77      0.73      166

accuracy          0.64
macro avg         0.63
weighted avg      0.64
    
```

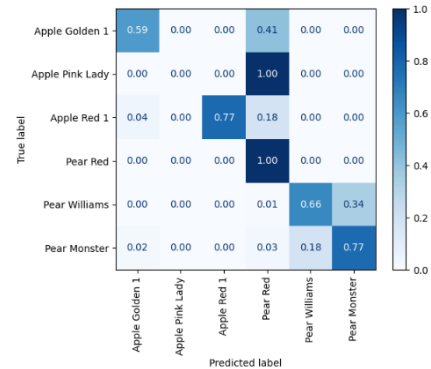


Fig. 32 Gráfico de las curvas de entrenamiento y validación después de entrenar, resultados evaluación del modelo y matriz de confusión del modelo

Por otro lado, al ejecutar las celdas del clasificador de noticias en las figuras 17, 18, 19 y 20, tendremos como resultados que el conjunto se cargó correctamente, las agrupaciones de las noticias, el número de ejemplos de cada conjunto, etc. Recordemos que son celdas que se enfocan en preparar los datos. Por otro lado, si ejecutamos la celda de las figuras 21, 22, 23 y 24, nos encontraremos con un resumen de nuestro modelo creado, el trazado del modelo, los resultados del entrenamiento y la representación gráfica que contiene las curvas de la pérdida y la precisión en el transcurso de las épocas (véase Fig. 33, 34, 35, 36).

```

Model: "sequential"
Layer (type)                Output Shape                Param #
-----
text_vectorization (TextVe  (None, None)               0
ctorization)
embedding (Embedding)       (None, None, 64)           640000
conv1d (Conv1D)              (None, None, 64)           20544
max_pooling1d (MaxPooling1  (None, None, 64)           0
D)
conv1d_1 (Conv1D)            (None, None, 64)           20544
global_max_pooling1d (Glob  (None, 64)                 0
alMaxPooling1D)
dense (Dense)                (None, 32)                  2080
dropout (Dropout)            (None, 32)                  0
dense_1 (Dense)              (None, 4)                   132

Total params: 683300 (2.61 MB)
Trainable params: 683300 (2.61 MB)
Non-trainable params: 0 (0.00 Byte)
    
```

Fig. 33 Resumen del modelo creado

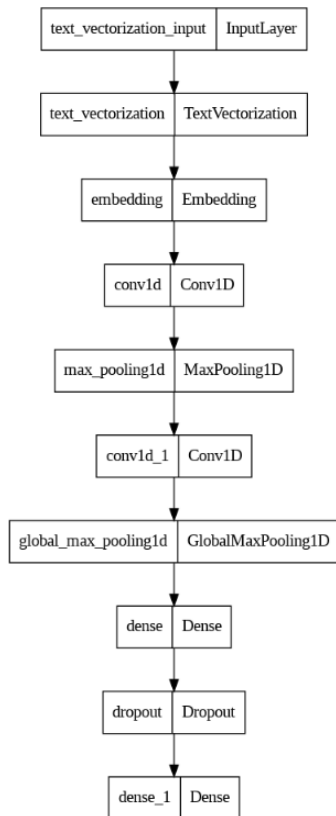


Fig. 34 Trazado del modelo creado

```

Epoch 1/25
105/105 [=====] - 23s 168ms/step - loss: 1.3722 - accuracy: 0.3018 - val_loss: 1.2521 - val_accuracy: 0.4901
Epoch 2/25
105/105 [=====] - 6s 76ms/step - loss: 0.9868 - accuracy: 0.5670 - val_loss: 0.6284 - val_accuracy: 0.7829
Epoch 3/25
105/105 [=====] - 2s 23ms/step - loss: 0.6375 - accuracy: 0.7753 - val_loss: 0.4551 - val_accuracy: 0.8583
Epoch 4/25
105/105 [=====] - 2s 22ms/step - loss: 0.4993 - accuracy: 0.8399 - val_loss: 0.4164 - val_accuracy: 0.8651
Epoch 5/25
105/105 [=====] - 2s 19ms/step - loss: 0.4398 - accuracy: 0.8562 - val_loss: 0.3724 - val_accuracy: 0.8799
Epoch 6/25
105/105 [=====] - 3s 23ms/step - loss: 0.4432 - accuracy: 0.8589 - val_loss: 0.3923 - val_accuracy: 0.8618
Epoch 7/25
105/105 [=====] - 2s 22ms/step - loss: 0.4012 - accuracy: 0.8735 - val_loss: 0.3644 - val_accuracy: 0.8799
Epoch 8/25
105/105 [=====] - 2s 17ms/step - loss: 0.4179 - accuracy: 0.8628 - val_loss: 0.3394 - val_accuracy: 0.8947
Epoch 9/25
105/105 [=====] - 1s 14ms/step - loss: 0.4023 - accuracy: 0.8714 - val_loss: 0.3463 - val_accuracy: 0.8865
Epoch 10/25
105/105 [=====] - 1s 10ms/step - loss: 0.3965 - accuracy: 0.8711 - val_loss: 0.3262 - val_accuracy: 0.8947
Epoch 11/25
105/105 [=====] - 1s 11ms/step - loss: 0.3787 - accuracy: 0.8842 - val_loss: 0.3146 - val_accuracy: 0.8947
Epoch 12/25
105/105 [=====] - 1s 12ms/step - loss: 0.3730 - accuracy: 0.8905 - val_loss: 0.3070 - val_accuracy: 0.8947
Epoch 13/25
105/105 [=====] - 1s 12ms/step - loss: 0.3763 - accuracy: 0.8756 - val_loss: 0.2853 - val_accuracy: 0.9112
Epoch 14/25
105/105 [=====] - 2s 17ms/step - loss: 0.3848 - accuracy: 0.8792 - val_loss: 0.2972 - val_accuracy: 0.9046
Epoch 15/25
105/105 [=====] - 1s 12ms/step - loss: 0.3826 - accuracy: 0.8833 - val_loss: 0.2727 - val_accuracy: 0.9112
Epoch 16/25
105/105 [=====] - 1s 11ms/step - loss: 0.3743 - accuracy: 0.8801 - val_loss: 0.2801 - val_accuracy: 0.9145
Epoch 17/25
105/105 [=====] - 1s 9ms/step - loss: 0.3635 - accuracy: 0.8851 - val_loss: 0.2702 - val_accuracy: 0.9145
Epoch 18/25
105/105 [=====] - 1s 8ms/step - loss: 0.3642 - accuracy: 0.8857 - val_loss: 0.2741 - val_accuracy: 0.9178
Epoch 19/25
105/105 [=====] - 1s 8ms/step - loss: 0.3517 - accuracy: 0.8827 - val_loss: 0.2681 - val_accuracy: 0.9178
Epoch 20/25
105/105 [=====] - 1s 8ms/step - loss: 0.3546 - accuracy: 0.8869 - val_loss: 0.2636 - val_accuracy: 0.9145
Epoch 21/25
105/105 [=====] - 1s 9ms/step - loss: 0.3483 - accuracy: 0.8905 - val_loss: 0.2681 - val_accuracy: 0.9178
Epoch 22/25
105/105 [=====] - 1s 11ms/step - loss: 0.3403 - accuracy: 0.8896 - val_loss: 0.2749 - val_accuracy: 0.9227
Epoch 23/25
105/105 [=====] - 1s 12ms/step - loss: 0.3042 - accuracy: 0.9033 - val_loss: 0.2668 - val_accuracy: 0.9211
Epoch 24/25
105/105 [=====] - 1s 11ms/step - loss: 0.3445 - accuracy: 0.8905 - val_loss: 0.2671 - val_accuracy: 0.9161
Epoch 25/25
105/105 [=====] - 2s 18ms/step - loss: 0.3370 - accuracy: 0.8914 - val_loss: 0.2688 - val_accuracy: 0.9079
  
```

Fig. 35 Resultados del entrenamiento del modelo

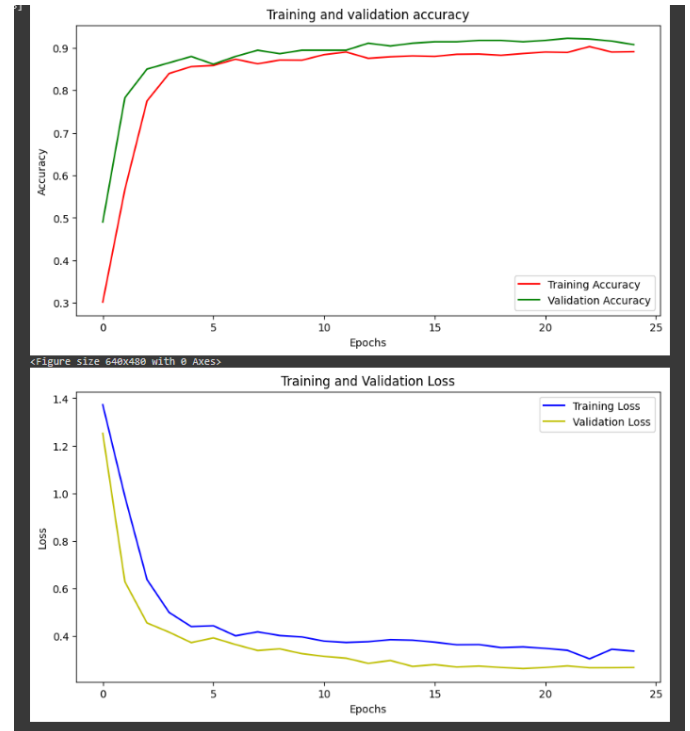


Fig. 36 Representación gráfica que contiene las curvas de la pérdida y la precisión en el transcurso de las épocas

Los últimos resultados que veremos de este clasificador son los de combinar el modelo de red neuronal convolucional con el modelo de red neuronal recurrente(CNN+RNN). Estos resultados los obtenemos al ejecutar las celdas de las figuras 25, 26 y 27. veremos un resumen de este modelo combinado, los resultados del entrenamiento, su representación gráfica y los resultados al probarlo(véase Fig. 37, 38, 39 y 40).

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
text_vectorization (TextVectorization)	(None, None)	0
embedding_1 (Embedding)	(None, None, 64)	640000
conv1d_2 (Conv1D)	(None, None, 64)	20544
max_pooling1d_1 (MaxPooling1D)	(None, None, 64)	0
conv1d_3 (Conv1D)	(None, None, 64)	20544
bidirectional_1 (Bidirectional)	(None, 128)	66048
dense_2 (Dense)	(None, 32)	4128
dropout_1 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 4)	132

=====  
 Total params: 751396 (2.87 MB)  
 Trainable params: 751396 (2.87 MB)  
 Non-trainable params: 0 (0.00 Byte)

Fig. 37 Resumen del modelo combinado

Epoch 1/25	145 87ms/step - loss: 1.3005 - accuracy: 0.3336 - val_loss: 0.9501 - val_accuracy: 0.5214
Epoch 2/25	145 145ms/step - loss: 0.7472 - accuracy: 0.7223 - val_loss: 0.4939 - val_accuracy: 0.8421
Epoch 3/25	145 145ms/step - loss: 0.5780 - accuracy: 0.8010 - val_loss: 0.4692 - val_accuracy: 0.8308
Epoch 4/25	145 145ms/step - loss: 0.4086 - accuracy: 0.8260 - val_loss: 0.3557 - val_accuracy: 0.8816
Epoch 5/25	145 145ms/step - loss: 0.4483 - accuracy: 0.8488 - val_loss: 0.3531 - val_accuracy: 0.8816
Epoch 6/25	145 145ms/step - loss: 0.4123 - accuracy: 0.8687 - val_loss: 0.3436 - val_accuracy: 0.8898
Epoch 7/25	145 145ms/step - loss: 0.4197 - accuracy: 0.8705 - val_loss: 0.3472 - val_accuracy: 0.8964
Epoch 8/25	145 145ms/step - loss: 0.3946 - accuracy: 0.8801 - val_loss: 0.3132 - val_accuracy: 0.8931
Epoch 9/25	145 145ms/step - loss: 0.3883 - accuracy: 0.8759 - val_loss: 0.3052 - val_accuracy: 0.8997
Epoch 10/25	145 145ms/step - loss: 0.3780 - accuracy: 0.8815 - val_loss: 0.3008 - val_accuracy: 0.8908
Epoch 11/25	145 145ms/step - loss: 0.3538 - accuracy: 0.8815 - val_loss: 0.3014 - val_accuracy: 0.8908
Epoch 12/25	145 145ms/step - loss: 0.3550 - accuracy: 0.8824 - val_loss: 0.2804 - val_accuracy: 0.8997
Epoch 13/25	145 145ms/step - loss: 0.3559 - accuracy: 0.8798 - val_loss: 0.2817 - val_accuracy: 0.9030
Epoch 14/25	145 145ms/step - loss: 0.3597 - accuracy: 0.8875 - val_loss: 0.2948 - val_accuracy: 0.8964
Epoch 15/25	145 145ms/step - loss: 0.3651 - accuracy: 0.8878 - val_loss: 0.2608 - val_accuracy: 0.9161
Epoch 16/25	145 145ms/step - loss: 0.3628 - accuracy: 0.8845 - val_loss: 0.2665 - val_accuracy: 0.9095
Epoch 17/25	145 145ms/step - loss: 0.3347 - accuracy: 0.8946 - val_loss: 0.2554 - val_accuracy: 0.9095
Epoch 18/25	145 145ms/step - loss: 0.3482 - accuracy: 0.8839 - val_loss: 0.2610 - val_accuracy: 0.9079
Epoch 19/25	145 145ms/step - loss: 0.3505 - accuracy: 0.8821 - val_loss: 0.2548 - val_accuracy: 0.9079
Epoch 20/25	145 145ms/step - loss: 0.3367 - accuracy: 0.8887 - val_loss: 0.2577 - val_accuracy: 0.9095
Epoch 21/25	145 145ms/step - loss: 0.3278 - accuracy: 0.8985 - val_loss: 0.2549 - val_accuracy: 0.9178
Epoch 22/25	145 145ms/step - loss: 0.3272 - accuracy: 0.8917 - val_loss: 0.2579 - val_accuracy: 0.9227
Epoch 23/25	145 145ms/step - loss: 0.3073 - accuracy: 0.9033 - val_loss: 0.2397 - val_accuracy: 0.9211
Epoch 24/25	145 145ms/step - loss: 0.3097 - accuracy: 0.9000 - val_loss: 0.2389 - val_accuracy: 0.9145
Epoch 25/25	145 145ms/step - loss: 0.3209 - accuracy: 0.8961 - val_loss: 0.2400 - val_accuracy: 0.9161

Fig. 38 Resultados entrenamiento del modelo combinado

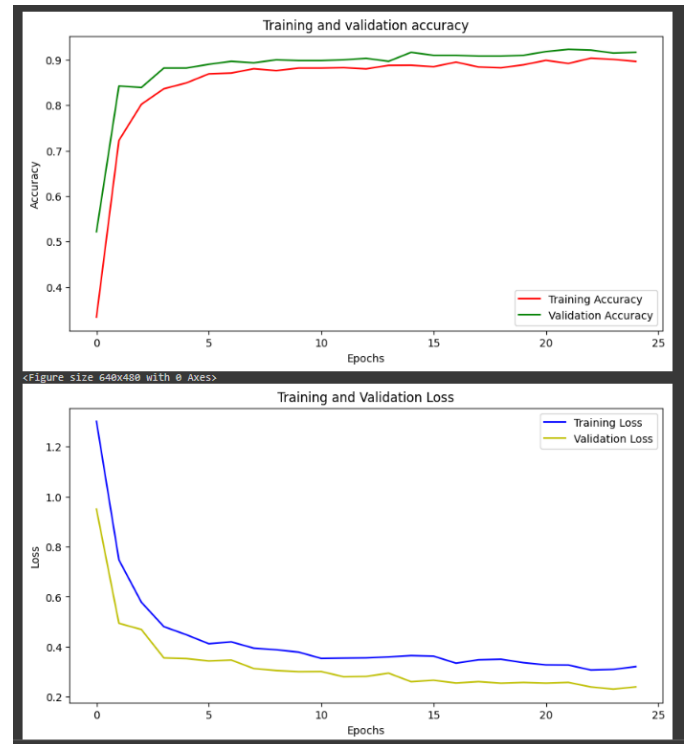


Fig. 39 Representación gráfica de los resultados del entrenamiento del modelo combinado

```
[30] sample_news = ["Tesla, a self-driving car company is also planning to make a humanoid robot. This humanoid robot appeared dancing in the latest Tesla AI day"]
predict(conv_rnn_model, sample_news, class_names)
1/1 [=====] - 1s 15/step
Predicted class: 3
Predicted class name: Sci/Tech

[31] sample_news = ["In the last weeks, there has been many transfer surprises in football. Ronaldo went back to old Trafford, "
                    "while Messi went to Paris Saint Germain to join his former colleague Neymar."
                    "We can't wait to see these two clubs will perform in upcoming leagues"]
predict(conv_rnn_model, sample_news, class_names)
1/1 [=====] - 0s 10ms/step
Predicted class: 1
Predicted class name: Sports

[32] sample_news = ["In the latest business news: The tech giant NVIDIA has acquired ARM, a microprocessor company"]
predict(conv_rnn_model, sample_news, class_names)
1/1 [=====] - 0s 10ms/step
Predicted class: 2
Predicted class name: Business
```

Fig. 40 Resultados de las pruebas hechas al modelo combinado

## V. ANALYSIS

Los análisis que podremos hacer gracias a los resultados obtenidos son los siguientes: Para empezar, observando los resultados del clasificador de frutas, uno de los primeros puntos a notar es que, analizando a grandes rasgos los resultados de la figura 31 y 32, existe un sobreajuste a partir de la epoch 6, a partir del cual se pierde de gran forma la precisión del conjunto de validación y aumenta su loss, mientras que estos mejoran ligeramente en test.

Ahora, entrando en más detalle dentro de los resultados obtenidos, como se puede ver en la última

figura obtenida de la figura 32, notamos que el modelo es capaz de identificar con una precisión de entre el 60 al 77% la mayoría de las frutas, pero con la Pear Red y Apple Pink Lady en particular, ambas fueron clasificadas el 100% del tiempo como pear red. Este fenómeno se observa a menor nivel con Apple Golden 1, y a mucho menor nivel con Apple Red 1, con lo cual se observa una tendencia del modelo a clasificar estas manzanas como Pear Red. A primera instancia, este problema es poco probable que ocurra debido a pocas muestras, ya que si bien Pear Red tiene más muestras que todos, no es una diferencia demasiado grande, teniendo aproximadamente 25% más muestras que el resto.

Con respecto a las otras frutas, tanto Pear Williams como Pear Monster tuvieron resultados prometedores, ya que si bien existía confusión entre estas, no se clasifican erróneamente en gran medida, siendo la Pear Monster la mejor clasificada de ambos. Otro resultado notable es que entre estas peras y el resto de frutas, no existe confusión, por lo que ambas presentan características que las diferencian fuertemente de un grupo a otro. Una posible razón podría ser el color y forma, ya que todas las manzanas son similares entre sí, y la Pear Red presenta un color similar a la mayoría, mientras que las otras dos peras, Williams y Monster, no comparten ni color ni forma con las demás, con la excepción de la forma con Pear Red.

Para el segundo caso, como se puede observar en la figura 35, el modelo ofrece resultados precisos, teniendo un buen rendimiento en el conjunto de validación y test. Sin embargo, puede existir un pequeño sobreajuste desde la epoch 22, ya que se empieza a perder precisión de validación, mientras que aumenta la del conjunto de test. Aún así, se observó este comportamiento, en menor medida, en epochs anteriores, y estos tendían a llevar una mejora después de más epochs a validación, por lo que podría también tratarse de un fenómeno temporal.

Ahora, con respecto al comportamiento del modelo a través de las epochs de su entrenamiento, su rendimiento mejora rápidamente hasta llegar a los 5 epochs, a partir de los cuales presenta mejoras, especialmente al disminuir su loss, pero lentamente. Aquí es donde puede notarse un poco mejor las caídas ocasionales de la precisión del modelo, y el lento incremento de ambos conjuntos. Para este caso, sería interesante poseer un tercer conjunto de noticias, y tras finalizar el entrenamiento, analizar el desempeño del modelo sobre

este, con el objetivo de analizar si ocurrió un sobreajuste, y si el modelo logra enfrentarse bien a estos nuevos datos.

Si bien los resultados indican un buen desempeño, siendo este superior al 90%, ahora, analizamos el resultado al incluir una RNN para generar la predicción de la categoría de la noticia, con tal de notar si existe una mejora sobre la clasificación. Como podemos ver en la figura 38, el desempeño obtenido es muy similar al modelo anterior, presentando fenómenos similares. En la figura 39 podemos ver que el comportamiento en el tiempo es bastante similar también, iniciando con un loss ligeramente menor. Sin embargo, el no hubo un incremento de rendimiento notable, a la vez que no hubo una pérdida de este tampoco. Esto es importante de considerar, ya que, en parte, el motivo de incluir un RNN es de mejorar el rendimiento del modelo.

Los resultados obtenidos indican que la inclusión del RNN no generó ni mejoras ni empeoramientos del desempeño del modelo. Con esto en mente, hay que considerar el aumento de la complejidad del modelo, al básicamente tener dos modelos en lugar de uno, lo cual sugiere la necesidad de buscar otras aproximaciones a estos problemas, ya que, considerando la cantidad de noticias que pueden llegar a existir, un 10% de error puede significar una cantidad consistente de noticias irrelevantes haciendo su aparición. Por otro lado, no se dispone de los resultados para cada clasificación, para analizar si estos modelos presentaban una precisión similar sobre todas sus categorías, o si estos presentaban comportamientos similares al modelo presentado en la última imagen de la figura 32, con pobre precisión en ciertas categorías y mejor en otras.

## VI. CONCLUSION

En base a los análisis realizados, se pudo observar un sobreajuste a partir del epoch(época) 6, este sobreajuste se ve reflejada en una pérdida significativa de precisión en el conjunto de validación, así también un aumento en la pérdida, mientras que, en el conjunto de prueba, los resultados mejoran ligeramente. Este comportamiento se ve más pronunciado en las clasificaciones de Pear Red y la manzana Pink Lady, que son clasificadas como Pear Red. Existe una tendencia a clasificar algunas manzanas como Pear Red, como Apple Golden y Apple Red 1 en menor medida, lo que puede significar una mayor

dificultad para el modelo para clasificar estas variedades.

Por otro lado, los resultados de Pear Williams y Pear Monster son más consistentes, con algunas confusiones entre ellas pero sin errores graves. Al haber menos errores entre las peras que en las manzanas, nos dice que al ser más distintivas entre sí (las peras), el modelo las puede clasificar de mejor manera. Otro factor a considerar es el color y la forma, ya que las manzanas comparten colores y formas similares, mientras que las peras son más diferentes entre sí.

En los análisis del clasificador de noticias, muestra un excelente rendimiento, se obtuvieron resultados precisos en los conjuntos de validación y prueba, aunque, existe la posibilidad de un pequeño sobreajuste a partir del epoch 22, además los resultados han demostrado algunas mejoras después de más epochs, el cual indica un fenómeno temporal. El rendimiento mejora desde el epoch 5 y luego experimenta débiles mejoras.

Luego se introdujo una RNN, el cual no presentó mejoras considerables en comparación al CNN, los resultados al ser un modelo más complejo, dieron resultados similares, lo que nos dice que se deben buscar otras alternativas para abordar este caso, ya que un 10% de error es significativo para las clasificaciones de noticias.

## REFERENCES

- [1] "Google Colaboratory". Google Colab. Accedido el 22 de diciembre de 2023. [En línea]. Disponible: [https://colab.research.google.com/drive/1FqF23ASvssd9RP5s-h96D\\_CHKMjVEopu?authuser=1#scrollTo=R3e0eEVg1xRe](https://colab.research.google.com/drive/1FqF23ASvssd9RP5s-h96D_CHKMjVEopu?authuser=1#scrollTo=R3e0eEVg1xRe)
- [2] "Google Colaboratory". Google Colab. Accedido el 22 de diciembre de 2023. [En línea]. Disponible: <https://colab.research.google.com/drive/1RD7HovvEcIcp2icUDFxW3n7nH6hN89S?authuser=1#scrollTo=Wx5hUh8ObHDC>
- [3] J. Rivas. "Redes Neuronales convolucionales y recurrentes". Medium. Accedido el 22 de diciembre de 2023. [En línea]. Disponible: [https://medium.com/@jirivas\\_73036/redes-neuronales-convolucionales-y-recurrentes-bd0c343802b7#:~:text=Las%20CNN%20\(redes%20neur onales%20convolucionales temporales,%20como%20texto%20o%20vi deos](https://medium.com/@jirivas_73036/redes-neuronales-convolucionales-y-recurrentes-bd0c343802b7#:~:text=Las%20CNN%20(redes%20neur onales%20convolucionales temporales,%20como%20texto%20o%20vi deos)
- [4] "Clasificación automática de frutas basada en IA". Ingivision. Accedido el 22 de diciembre de 2023. [En línea]. Disponible: <https://www.ingivision.com/2023/03/22/clasificacion-automatica-frutas-ia/>
- [5] Paloma Recuero de los Santos. "Cómo interpretar la matriz de confusión: ejemplo práctico". Telefónica Tech. Accedido el 28 de octubre de 2023. [En línea]. Disponible: <https://telefonicatech.com/blog/como-interpretar-la-matriz-de-confusion-ejemplo-practico#:~:text=La%20matriz%20de%20confusión%20es,c on%20distintos%20tipos%20de%20error>
- [6] Skills Tech. Demo Day Generación 7 | Detector y Clasificador de Noticias Falsas por Christian Ángeles. (14 de marzo de 2023). Accedido el 22 de diciembre de 2023. [Video en línea]. Disponible: [https://www.youtube.com/watch?v=xwZ8H407\\_LA](https://www.youtube.com/watch?v=xwZ8H407_LA)