Integrantes:
Simón Porras - Juanita Palacios

Proyecto - Corte 1

1. Criterios de diseño del modelo 1

Previamente, se asignó una arquitectura Multilayer Perceptron (MLP) por indicaciones previas, y el dataset de MedMNIST, ChestMNIST, que consiste en imágenes en escala de grises con las que se trabajó un tamaño 28x28 píxeles y se realiza una clasificación multiclase (multi-label:14 según MedMNIST) con un modelo secuencial TensorFlow

Posteriormente, se utilizó la distribución predeterminada del dataset y se distribuyó el conjunto de datos, con su posterior comprobación:

```
Train shape: (78468, 28, 28, 1)
Validation shape: (11219, 28, 28, 1)
Test shape: (22433, 28, 28, 1)
```

- **x_train (tran shape):** 78,468 imagenes de 28x28 px en escala de grises (1 canal)
- **x_val (validation shape):** 11,219 imágenes por validació
- **x_test (test shape):** 22,433 imagenes para prueba final

En la capa de salida se utilizaron 14 neuronas (una por cada etiqueta), con activación sigmoid, que devuelve una probabilidad entre 0 y 1 para cada clase individualmente. Para evitar sobreajuste, se aplicaron capas Dropout, y relu como función de activación en las capas ocultas.

```
model = tf.keras.Sequential([

    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),

    tf.keras.layers.Dense(512, activation='relu'), # capa Dense con tantas neuronas como
clases posibles

    tf.keras.layers.Dropout(0.3),

 tf.keras.layers.Dense(256, activation='relu'),

    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Dense(128, activation='relu'), #tf.keras.layers.Dense(64,
activation='relu'),

    tf.keras.layers.Dense(14, activation='sigmoid')  # 3 etiquetas multilabel

])
```

- Función de pérdida: binary_crossentropy, debido al multi-label del dataset
- Épocas: 20, Batch size: 128, optimizador: Adam (con tasa de aprendizaje 0,1); variables dispuestas teniendo en cuenta referencias vistas en clase

1.1 Características del modelo (model.summary)

La siguiente tabla (incluida en el código) representa en sus columnas respectivamente: tipo de capa, unidades y cantidad de parámetros

Model: "sequential_10"

Layer (type)	Output Shape	Param #
flatten_10 (Flatten)	(None, 784)	0
dense_31 (Dense)	(None, 512)	401,920
dropout_9 (Dropout)	(None, 512)	0
dense_32 (Dense)	(None, 256)	131,328
dropout_10 (Dropout)	(None, 256)	0
dense_33 (Dense)	(None, 128)	32,896
dense_34 (Dense)	(None, 14)	1,806

Total params: 1,703,852 (6.50 MB)
Trainable params: 567,950 (2.17 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,135,902 (4.33 MB)

2. Resultados de la selección de hiperparámetros y generalización.

Los hiperparámetros fueron establecidos manualmente, teniendo en cuenta los criterios dados en clase anteriormente, de modo que se dispone de:

- Optimizador Adam, tasa de aprendizaje 0.1
- Batch size: 128
- Épocas: 20
- Dropout rate: 0.3
- Número de neuronas por capa: 512, 256, 128

```
# Hiperparámetros
batch_s = 128
lr = 0.1
num_epochs = 20
optimizerf = tf.keras.optimizers.Adam(learning_rate=lr)
```

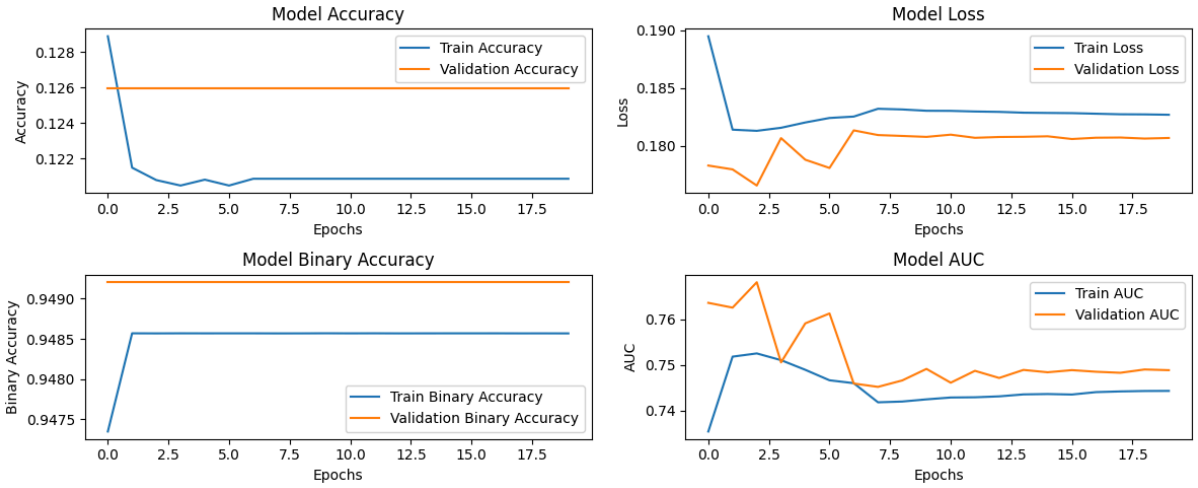
Con los hiperparámetros seleccionados, se escogieron tres métricas de evaluación para determinar el rendimiento del modelo:

- Accuracy: aunque es una métrica ampliamente utilizada, en problemas multilabel puede resultar engañosa, ya que exige que todas las etiquetas de una muestra sean clasificadas correctamente para considerarla como un acierto.

- Binary Accuracy: mide el porcentaje de aciertos por etiqueta individual, promediando sobre todas las salidas y muestras. Esta métrica resulta más adecuada para evaluar modelos multilabel, dado que ofrece una visión más realista del rendimiento.

- AUC (Área bajo la curva ROC): utilizada para medir la capacidad de discriminación del modelo. En el contexto multilabel, se calcula de manera independiente por cada etiqueta y luego se promedia, ofreciendo información sobre qué tan bien el modelo separa clases positivas y negativas.

Las gráficas de entrenamiento y validación para estas métricas se presentan a continuación:



La métrica accuracy se mantuvo baja y mostró una curva estable en validación, lo cual confirma que no es la mejor métrica para este tipo de problema. La binary accuracy se mantuvo estable y alta, aunque puede estar influida por el desbalance en las etiquetas. El AUC mostró una tendencia positiva en entrenamiento y validación, siendo la métrica más representativa del aprendizaje y la generalización del modelo.

3. Evaluación de rendimiento.

Por último con ese modelo se realizó una evaluación de rendimiento con la parte de pruebas del dataset usando la función evaluate() del modelo. Dando como resultado lo

siguiente:

```
# Evaluar el modelo en el conjunto de prueba
test_loss, test_binary_acc, test_acc, test_auc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {test_acc:.4f} - AUC: {test_auc:.4f} - Binary Accuracy: {test_binary_acc:.4f}")
```

✓ 1.7s

```
702/702 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.1204 - auc_7: 0.7458 - binary_accuracy: 0.9474 - loss: 0.1855

Test Accuracy: 0.1204 - AUC: 0.7458 - Binary Accuracy: 0.9474
```

Los resultados obtenidos en el conjunto de prueba evidencian las particularidades de la evaluación en problemas de clasificación multilabel. La métrica de accuracy alcanzó un valor de 0.1204, lo cual resulta bajo debido a que este indicador exige la coincidencia exacta de todas las etiquetas de una muestra, volviéndose poco representativo en escenarios donde cada instancia posee múltiples salidas posibles. En contraste, la binary accuracy registró un valor elevado (0.9474), lo cual refleja un alto nivel de acierto en la predicción individual de etiquetas; sin embargo, esta métrica puede estar influida por el desbalance existente en el dataset, ya que tiende a sobreestimar el rendimiento cuando las clases negativas son predominantes. Finalmente, el AUC (0.7458) se presenta como la métrica más informativa para este caso, dado que mide la capacidad discriminativa del modelo entre clases positivas y negativas, mostrando un desempeño aceptable y evidenciando que el modelo ha logrado aprender patrones relevantes que generalizan más allá del conjunto de entrenamiento.

4. Criterios de diseño del modelo 2

- En cuanto a las especificaciones generales del dataset y la arquitectura del modelo, se mantuvieron las mismas condiciones previamente descritas en el Modelo 1. La principal diferencia en esta segunda experimentación radicó en la incorporación de Keras Tuner como herramienta para la optimización de hiperparámetros. A través de este proceso se exploraron y seleccionaron automáticamente configuraciones relacionadas con las funciones de activación, la cantidad de capas ocultas y las funciones de pérdida empleadas en la red. El objetivo de esta búsqueda fue identificar la combinación de parámetros que permitiera mejorar el rendimiento del modelo, ajustando su complejidad y capacidad de aprendizaje de manera más eficiente. La arquitectura definida a seguir para el tuner se presenta a continuación:

```
def build_model(hp):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28, 1)))

    # Número de capas ocultas
    for i in range(hp.Int('num_layers', 1, 3)):
        model.add(tf.keras.layers.Dense(
            units=hp.Choice('units_' + str(i), [128, 256, 512]),
            activation=hp.Choice('activation_' + str(i), ['relu', 'tanh', 'sigmoid'])
        ))
        model.add(tf.keras.layers.Dropout(rate=hp.Float('dropout_' + str(i), 0.1, 0.5, step=0.1)))

    model.add(tf.keras.layers.Dense(14, activation='sigmoid'))  # Salida multi-label

    model.compile(
        optimizer=hp.Choice('optimizer', ['adam', 'rmsprop', 'sgd']),
        loss='binary_crossentropy',
        metrics=['binary_accuracy','accuracy', tf.keras.metrics.AUC]
    ) #

    return model
```

- En la optimización de la arquitectura con Keras Tuner se establecieron los siguientes criterios de búsqueda:
    - Capas ocultas
        - Cantidad: entre 1 y 3.
        - Tipo: Densas (fully connected).
    - Parámetros de cada capa oculta
        - Número de neuronas: 128, 256, 512.
        - Funciones de activación: ReLU, Sigmoid, TanH.
    - Regularización
        - Después de cada capa densa se añadió una capa Dropout.
        - Valor de la tasa de desactivación: entre 0.1 y 0.5.
    - Optimizadores
        - Se consideraron tres opciones: Adam, RMSProp y SGD.
- Luego se procedió a realizar la búsqueda del mejor modelo, este se ajustó para encontrar el modelo que tuviera el mejor accuracy de validación durante 5 ensayos.

4.1 Características del modelo (model.summary)

La siguiente tabla (incluida en el código) representa en sus columnas respectivamente: tipo de capa, unidades y cantidad de parámetros del seleccionado como mejor modelo luego de realizar la búsqueda.

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 256)	200,960
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 14)	3,598

Total params: 409,118 (1.56 MB)

Trainable params: 204,558 (799.05 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 204,560 (799.07 KB)

5. Resultados de la selección de hiperparámetros y generalización.

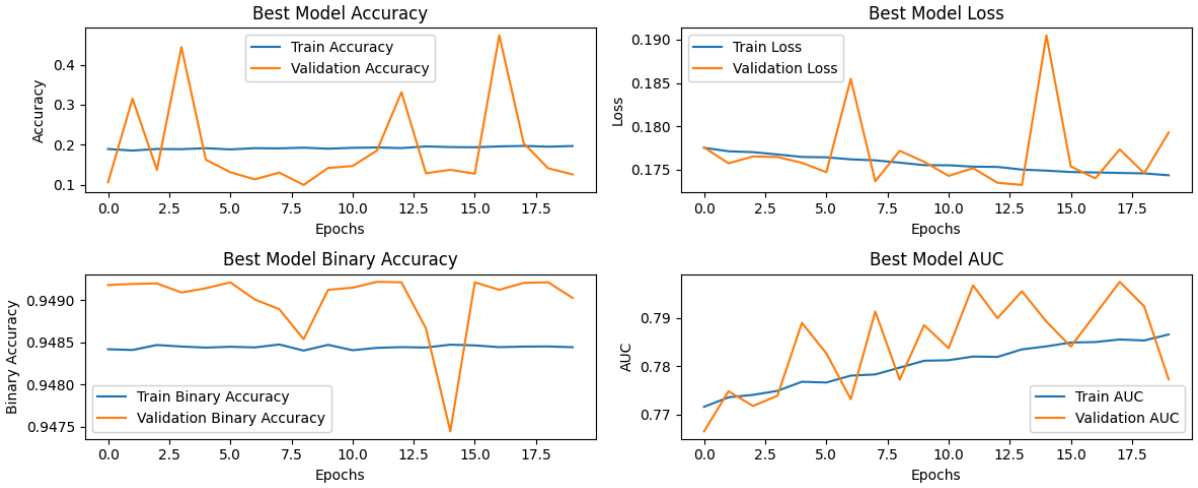
En este caso, algunos hiperparámetros fueron definidos manualmente, mientras que otros dependieron de la exploración realizada por Keras Tuner. Los valores finales seleccionados fueron los siguientes:

- Hiperparámetros definidos manualmente:
    - Tamaño de lote (batch size): 128.
    - Número de épocas (epochs): 20.
- Hiperparámetros seleccionados por el tuner:
    - Optimizador: RMSProp.
    - Learning rate: aproximadamente 0.001.
    - Capas densas: 1 capa con 256 neuronas.

- Función de activación: TanH.
- Capa de regularización **(Dropout):** tasa de desactivación de **0.5**.

```
Optimizer: rmsprop - Lr: 0.0010000000474974513
Activation: tanh
Dropout: 0.5
```

Con el modelo ya escogido se procedió a determinar el rendimiento del entrenamiento, usando las métricas ya mencionadas anteriormente, dando como resultado las siguientes gráficas:



La métrica accuracy se mantuvo baja y mostró variaciones considerables en validación, lo cual confirma que no es la mejor métrica para este tipo de problema. La binary accuracy se mantuvo estable y alta en entrenamiento, pero en validación se pueden observar algunos picos. El AUC mostró una tendencia positiva en entrenamiento y validación.

6. Evaluación de rendimiento.

Por último se procedió a realizar la evaluación de rendimiento de igual forma que con el modelo anterior con la función evaluate(). Arrojando los siguientes resultados:

```
Mejor modelo: Accuracy: 0.2188 - AUC: 0.7831 - Binary Accuracy: 0.9474
```

Los valores de rendimiento muestran una mejoría en la métrica de accuracy con respecto al modelo manual, aunque, como se mencionó anteriormente, esta métrica no resulta adecuada para problemas de tipo multilabel debido al desbalance de clases. Por otro lado, la métrica de AUC también presentó una ligera mejora en comparación con el modelo manual, lo que la convierte en la más informativa y confiable para evaluar el desempeño. En este sentido, se puede evidenciar que el uso del tuner contribuyó a encontrar una configuración de hiperparámetros que permitió un mejor ajuste del modelo y una mayor capacidad de generalización.

6. Link repositorio del proyecto.

https://github.com/JPalacios-8/DL-Proyecto-1/blob/main/project2.ipynb