INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Maestría en Inteligencia Artificial Aplicada (MNA)

Proyecto Integrador (TC5035)

Dra. Grettel Barceló Alonso

Dr. Luis Eduardo Falcón Morales

Asesora: Dra. María de la Paz Rico Fernánde

Avance 3

Baseline

“Clasificación de ruido en laboratorio de motores eléctricos automotrices a través de métodos de inteligencia artificial”

EQUIPO 14

Andrei García Torres A01793891
Emmanuel González Calitl A01320739
Denisse María Ramírez Colmenero A01561497

Fecha: 19 de mayo de 2024

Este proyecto consiste en la creación de un modelo capaz de clasificar imágenes de espectrogramas de ruido de motores elevadores en ruido aceptable (1 - ok) o rechazadas (0 - nok) con el fin de lograr la detección temprana de ruido y así mejorar la calidad del producto que será entregado a un cliente final. Estas imágenes han sido etiquetadas previamente como tal por el departamento de ruido de la planta de Bosch México en Toluca por medio de equipos especializados en la medición de ruido.

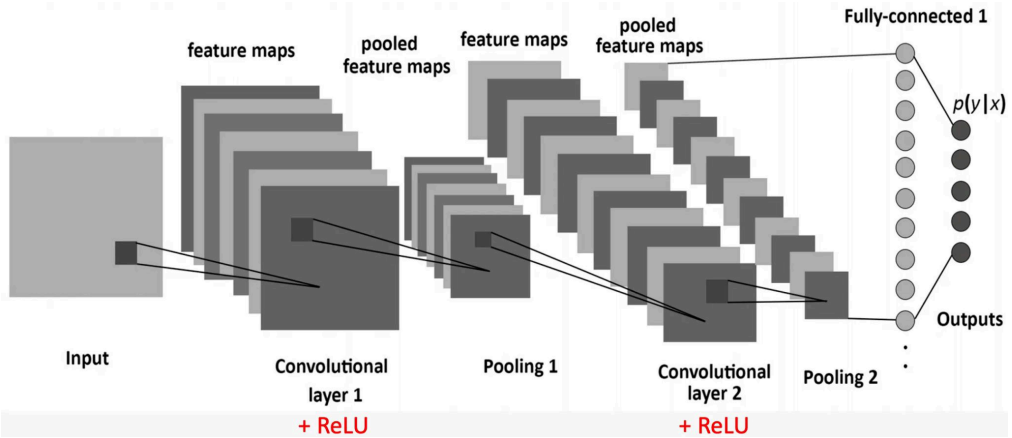
Para lograr un modelo de clasificación exitoso es necesario contar con un Baseline como punto de referencia con el objetivo de proporcionar un métrica inicial para comparar el rendimiento de los siguientes modelos mas avanzados que el equipo realizará y así poder evaluar si estos modelos complejos en realidad tienen un mejor desempeño en comparación con un modelo simple antes de la creación de uno de estos. La construcción de un baseline también ayuda a identificar si hay algún problema con los datos de entrada o con el entrenamiento.

Para el algoritmo de aprendizaje automático para la clasificación de los espectrogramas se utilizarán redes neuronales convolucionales (CNN) las cuales están orientadas a capturar información espacial de las imágenes a través de filtros bidimensionales.

Existen tres tipos de capas para una CNN:

- Convolucional
- Pooling
- Fully-connected

Figura 1.
Capas de una red neuronal convolucional

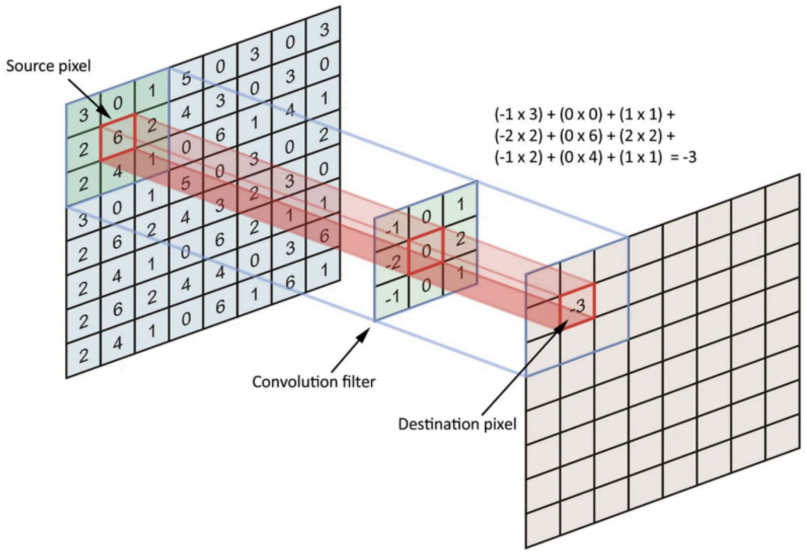


La **capa convolucional** se basa en la operación de convolución y un kernel 2D, también llamado filtro. Los filtros se aplican para extraer características y crear mapas de características (feature maps).

Los parámetros son:

- Número de kernels
- Tamaño de activación
- Función de activación
- Stride
- Padding

Figura 2.
Convolución con un Kernel 3x3



Nota. El kernel o filtro se desliza sobre la imagen de entrada y calcula una suma ponderada de los valores de los píxeles produciendo una matriz nueva llamada mapa de características.

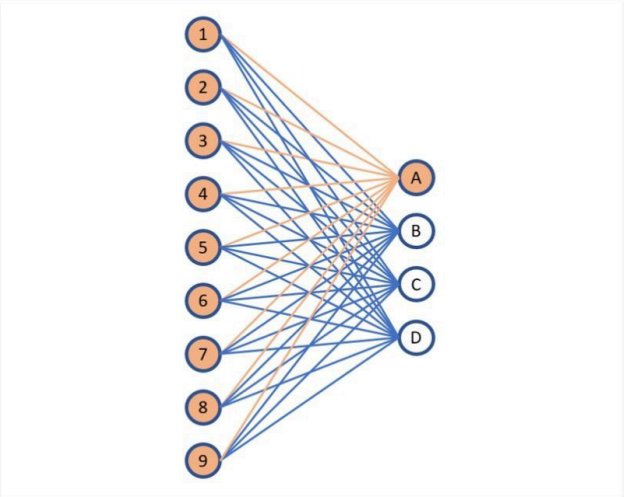
Las **capas Pooling** usualmente proceden de las capas convolucionales y reducen la dimensionalidad de la información recorriendo los features maps. Esto ayuda a conservar las características más importantes y reducir el costo computacional. Los parámetros son:

- Stride
- Tamaño de ventana

Las **capas Fully-connected** usualmente preceden a la clasificación. En una clasificación, su objetivo es aplanar (flatten) los resultados de las capas de convolución o pooling en un vector unidimensional, pasarlas por las neuronas fully-connected y lograr una clasificación donde la red aprenda relaciones complejas. Las neuronas de estas capas están conectadas a todas las neuronas de la capa anterior, como se muestra en la figura 3. Aquí los parámetros son:

- Número de nodos
- Función de activación

Figura 3.
Vector unidimensional con capa fully-connected

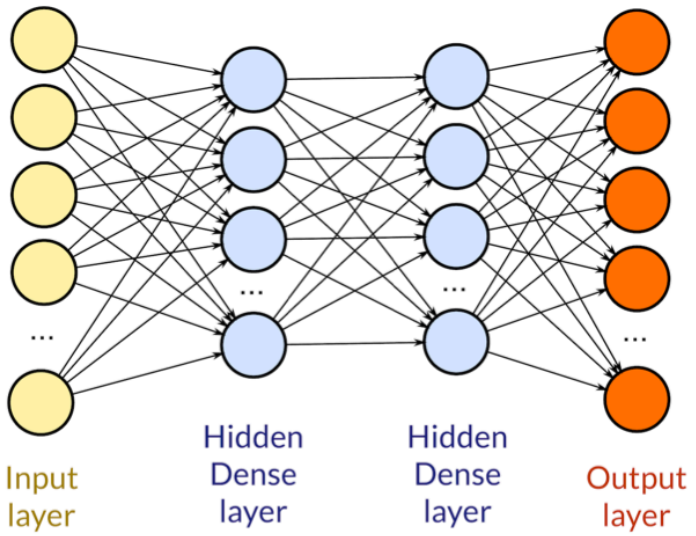


Algoritmo Baseline

Como punto de referencia, el equipo de trabajo ha construido un primer modelo de red neuronal densa utilizando la API “Sequential” de Keras para realizar el primer acercamiento para la clasificación de los espectrogramas de ruido OK y NOK.

Las redes densas son las capas más simples de las redes neuronales, donde cada neurona está conectada a cada neurona de la capa anterior. Los valores de salida se calculan multiplicando cada valor de salida de la capa anterior con el peso de conexión de cada neurona. (Verdeguer J, 2020).

Figura 4.
Capas de una red neuronal densa



Nota. Tomada de la Tesis de maestría “Redes neuronales para la clasificación y segmentación de imágenes médicas”, Verdeguer J, 2020, Universitat Politècnica de València.

Nuestro baseline de redes densas utiliza una capa inicial de 50 neuronas, dos capas de 10 neuronas cada una y capa de salida de una neurona, debido al número de clases que tenemos. La dimensión de entrada en la primera capa es de 100,000 (debido a que es el valor de ancho por alto de las imágenes, es decir, 250 x 400).

```
model = Sequential()
model.add(Dense(units = 50, input_dim=num_pixels, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(units = 10, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(units = 10, activation='softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
print(model.summary())
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usi
ng an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 50)	5,000,050
dense_1 (Dense)	(None, 10)	510
dense_2 (Dense)	(None, 10)	110

Total params: 5,000,670 (19.08 MB)

Trainable params: 5,000,670 (19.08 MB)

Non-trainable params: 0 (0.00 B)

None

A continuación se presenta el código de la construcción del modelo red neuronal densa para la clasificación de imágenes de espectrogramas de ruido utilizando el algoritmo que se describió anteriormente.

El archivo .ipynb se puede encontrar en el repositorio de Github en el link: https://github.com/emm-gl/project_mna/blob/main/Clasificaci%C3%B3n_Keras_20240516/Keras_Noise_2.2.ipynb

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
#from keras.utils.np_utils import to_categorical

import random

from keras.layers import Dropout
from keras.layers import Flatten

from tensorflow.keras.layers import Conv2D
#from keras.layers.convolutional import Conv2D

from tensorflow.keras.layers import MaxPooling2D
#from keras.layers.convolutional import MaxPooling2D

from keras.models import Model

import matplotlib.image as mpimg
import seaborn as sns
import cv2

#from google.colab import drive
import glob
```

In [3]:
```
file_path = 'Desktop/proyecto integrador/*'
```

In [4]:
```
noise = glob.glob("OK_etiquetadas_500-2/*.png")

nonoise = glob.glob("NOK_etiquetadas_500-2/*.png")
```

In [5]:
```
len(noise)
```

Out[5]:
```
500
```

In [6]:
```
len(nonoise)
```

Out[6]:
```
500
```

In [7]:
```
noise_Formated = []
nonoise_Formated = []

width = 400
height = 250
dim = (width, height)

for i in noise:
    img_gray = mpimg.imread(i)
    #resize image
    img_resized = cv2.resize(img_gray, dim, interpolation = cv2.INTER_AREA)

    #add the new image to a new array
    noise_Formated.append(img_resized)

for i in nonoise:
    img_gray = mpimg.imread(i)
    #resize image
    img_resized = cv2.resize(img_gray, dim, interpolation = cv2.INTER_AREA)

    #add the new image to a new array
    nonoise_Formated.append(img_resized)
```

In [8]:
```
y_noise = np.ones(len(noise_Formated))
y_nonoise = np.zeros(len(nonoise_Formated))
```

In [9]:
```python
X = np.vstack((noise_Formated,nonoise_Formated))
X.shape
```

Out[9]:
```
(1000, 250, 400)
```

In [10]:
```python
y = np.hstack((y_noise,y_nonoise))
y.shape
```

Out[10]:
```
(1000,)
```

In [11]:
```python
X=X/255
```

In [12]:
```python
# 400 x 250 = 100,000
num_pixels = 100000
X = X.reshape(X.shape[0], num_pixels)
print(X.shape)
```

```
(1000, 100000)
```

In [13]:
```python
model = Sequential()
model.add(Dense(units = 50, input_dim=num_pixels, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(units = 10, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(units = 10, activation='softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate=0.1)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
print(model.summary())
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usi
ng an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	**Output Shape**	**Param #**
dense (Dense)	(None, 50)	5,000,050
dense_1 (Dense)	(None, 10)	510
dense_2 (Dense)	(None, 10)	110

Total params: 5,000,670 (19.08 MB)

Trainable params: 5,000,670 (19.08 MB)

Non-trainable params: 0 (0.00 B)

None

In [14]:
```python
print(X.shape)
print(y.shape)
print(y)
```

```
(1000, 100000)
(1000,)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
# 25% para el set de prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 42)
```

In [16]:
```
model = Sequential()
model.add(Dense(units=1, input_dim=100000, activation='sigmoid'))
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy',  metrics=['accuracy', 'precision', 'r
```

```
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usi
ng an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [17]:
```
hist = model.fit(x=X_train, y=y_train, verbose=2, batch_size=32, epochs=100,
                 validation_data=(X_test, y_test), shuffle='true')
```

```
0 - val_loss: 0.6185 - val_precision: 0.6071 - val_recall: 0.6967
Epoch 95/100
24/24 - 0s - 13ms/step - accuracy: 0.7093 - loss: 0.5616 - precision: 0.7062 - recall: 0.7249 - val_accuracy: 0.636
0 - val_loss: 0.6184 - val_precision: 0.6099 - val_recall: 0.7049
Epoch 96/100
24/24 - 0s - 16ms/step - accuracy: 0.7107 - loss: 0.5611 - precision: 0.7069 - recall: 0.7275 - val_accuracy: 0.636
0 - val_loss: 0.6183 - val_precision: 0.6099 - val_recall: 0.7049
Epoch 97/100
24/24 - 0s - 13ms/step - accuracy: 0.7187 - loss: 0.5608 - precision: 0.7114 - recall: 0.7434 - val_accuracy: 0.636
0 - val_loss: 0.6183 - val_precision: 0.6099 - val_recall: 0.7049
Epoch 98/100
24/24 - 0s - 13ms/step - accuracy: 0.7120 - loss: 0.5603 - precision: 0.7077 - recall: 0.7302 - val_accuracy: 0.636
0 - val_loss: 0.6180 - val_precision: 0.6099 - val_recall: 0.7049
Epoch 99/100
24/24 - 0s - 18ms/step - accuracy: 0.7133 - loss: 0.5597 - precision: 0.7084 - recall: 0.7328 - val_accuracy: 0.636
0 - val_loss: 0.6180 - val_precision: 0.6099 - val_recall: 0.7049
Epoch 100/100
24/24 - 0s - 16ms/step - accuracy: 0.7147 - loss: 0.5592 - precision: 0.7092 - recall: 0.7354 - val_accuracy: 0.636
0 - val_loss: 0.6178 - val_precision: 0.6099 - val_recall: 0.7049
```

In [18]:
```
# resumen del model
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 1)	100,001

Total params: 300,005 (1.14 MB)

Trainable params: 100,001 (390.63 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 200,004 (781.27 KB)

In [22]:
```
# Obtén los datos del historial
accuracy = hist.history['accuracy']
test_accuracy = hist.history['val_accuracy']

precision = hist.history['precision']
test_precision = hist.history['val_precision']

recall = hist.history['recall']
test_recall = hist.history['val_recall']

loss = hist.history['loss']
test_loss = hist.history['val_loss']


# Crea la figura y los subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(8, 6))
```

```
# Gráfica de accuracy
ax1.plot(accuracy, label='Accuracy Entrenamiento')
ax1.plot(test_accuracy, label='Accuracy Prueba')
ax1.set_xlabel('Épocas')
ax1.set_ylabel('Accuracy')
ax1.legend()

# Gráfica de precision
ax2.plot(precision, label='Precision Entrenamiento')
ax2.plot(test_precision, label='Precision Prueba')
ax2.set_xlabel('Épocas')
ax2.set_ylabel('Precision')
ax2.legend()

# Gráfica de recall
ax3.plot(recall, label='Recall Entrenamiento')
ax3.plot(test_recall, label='Recall Prueba')
ax3.set_xlabel('Épocas')
ax3.set_ylabel('Recall')
ax3.legend()

# Gráfica de pérdida
ax4.plot(loss, label='Pérdida Entrenamiento')
ax4.plot(test_loss, label='Pérdida Prueba')
ax4.set_xlabel('Épocas')
ax4.set_ylabel('Pérdida')
ax4.legend()

# Ajusta los márgenes y espacios entre subplots
plt.tight_layout()

# Muestra las gráficas
plt.show()
```

Características importantes

Para la generación de este modelo simple, se utilizó el conjunto de datos al cual se le realizó un procedimiento de data augmentation anteriormente para así lograr un conjunto de clases balanceadas con 500 imágenes cada una. Cada una de estas imágenes fue renombrada con un "ok" o "nok" seguido de un ID. Una vez teniendo conjunto de datos de 1000 imágenes, estas fueron convertidas a escala de grises y redimensionadas para aportar simplicidad al modelo.

In [7]:
```
noise_Formated = []
nonoise_Formated = []

width = 400
height = 250
dim = (width, height)

for i in noise:
    img_gray = mpimg.imread(i)
    #resize image
    img_resized = cv2.resize(img_gray, dim, interpolation = cv2.INTER_AREA)

    #add the new image to a new array
    noise_Formated.append(img_resized)

for i in nonoise:
    img_gray = mpimg.imread(i)
    #resize image
    img_resized = cv2.resize(img_gray, dim, interpolation = cv2.INTER_AREA)

    #add the new image to a new array
    nonoise_Formated.append(img_resized)
```

Sub/sobreajuste

Respecto a la última época, se obtuvieron los siguientes resultados:

Tabla 1.

Resultados de entrenamiento y prueba.

	Accuracy	Precision	Recall	Loss
Entrenamiento	0.7147	0.7092	0.7354	0.5592
Prueba	0.630	0.6099	0.7049	0.6178

Métricas de desempeño

Como se mencionó anteriormente se utilizó un conjunto balanceado donde la clase 0 representa los ruidos de motores catalogados como NOK (no aceptables), mientras que la clase 1 representa los ruidos de motores OK, es decir, aceptables.

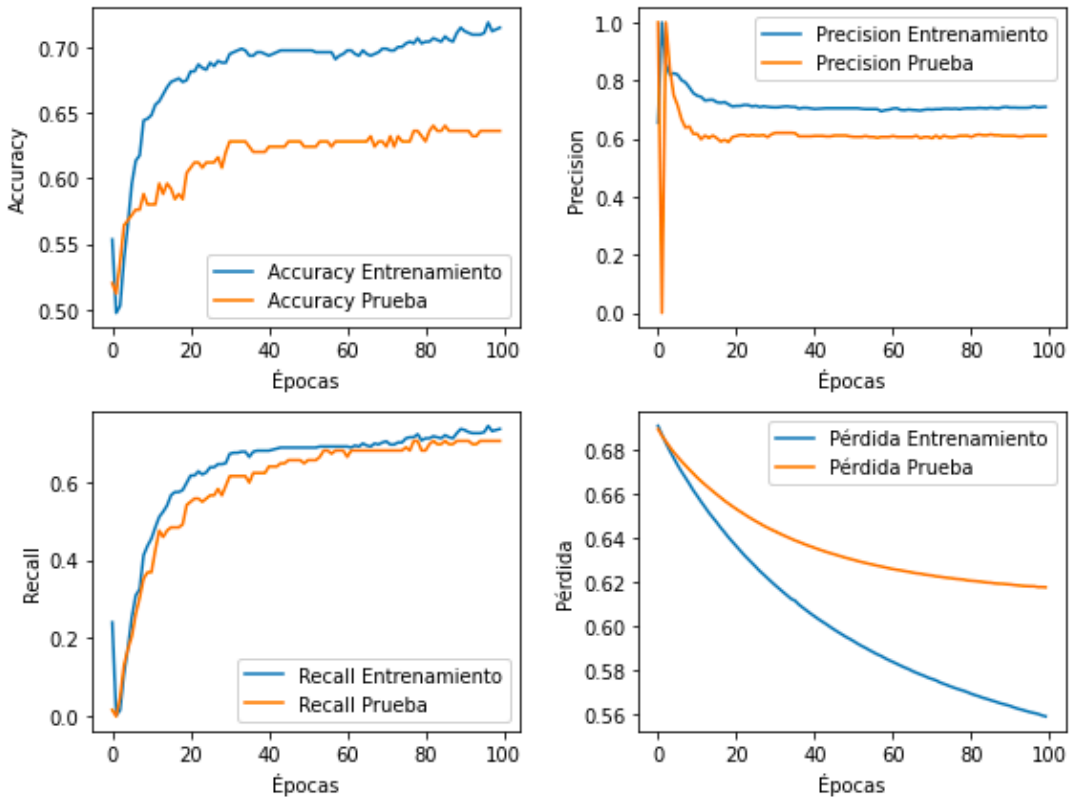
En este proyecto, tener un porcentaje bajo de falsos positivos es crucial, debido a que no queremos que un motor con un ruido no aceptable sea catalogado incorrectamente como aceptable, pues traerá consecuencias negativas a la empresa al permitir el uso de un motor con ruido inaceptable en la integración del producto final. Por otro lado, el tener poca cantidad de falsos negativos también es crucial, pues que un motor con ruido aceptable sea catalogado como no aceptable, resulta en el descarte de motor potencialmente bueno.

Teniendo en cuenta estos aspectos, y habiendo obtenido resultados muy similares en las distintas métricas, para nuestro baseline de red neuronal densa se seleccionó la métrica de exactitud o **accuracy** para evaluar el desempeño del modelo realizado. La métrica de accuracy es útil cuando se tiene un balance de clases y el peso de los errores de la clasificación (FP y FN) es similar. El accuracy se calcula dividiendo el número de predicciones correctas entre el total de predicciones.

Además se utilizó la métrica de pérdida o **Loss** para medir el error durante el entrenamiento y reducir la diferencia entre las predicciones y los valores reales. Un buen

modelo debe tener una disminución constante de la pérdida. Se utilizó la función Cross-Entropy debido a que es la más común para modelos de clasificación binaria.

Figura 5.
Métricas de desempeño del modelo propuesto en el Baseline



Conclusiones:

- El modelo muestra signos de overfitting. Esto se evidencia por el rendimiento significativamente mejor en el conjunto de entrenamiento en comparación con el conjunto de prueba en todas las métricas (accuracy, precisión, recall y pérdida).
- Para mejorar la generalización del modelo, se podrían considerar técnicas como el uso de regularización (L2, dropout), aumentar la cantidad de datos de entrenamiento mediante data augmentation o mejorar este algoritmo, o emplear métodos de early stopping durante el entrenamiento para evitar sobre ajustar los datos de entrenamiento.

Bibliografía

Bulentsiyah. (2019, 12 enero). *Dogs vs. Cats Classification (VGG16 Fine Tuning)*. Kaggle. https://www.kaggle.com/code/bulentsiyah/dogs-vs-cats-classification-vgg16-fine-tuning

Chávez, S. R. (2024, 9 mayo). Teaching a Machine to Learn Command Voices. https://www.linkedin.com/pulse/teaching-machine-learn-command-voices-santiago-reyes-ch%2525C3%2525A1vez-vm2cc/?trackingId=UAoFhFMITAeq479XXyb15A%3D%3D

Francois Chollet. (2021). Deep Learning with Python, Second Edition. Manning.

Perez, R. (2023, 29 noviembre). Clasificación de imágenes con redes profundas - Raul Perez - Medium. Medium. https://medium.com/@raulpzs/clasificaci%C3%B3n-de-im%C3%A1genes-con-redes-profundas-3a3b747489dc

Prathap, P. (2023, 28 julio). The Secret to Understanding CNNs: Convolution, Feature Maps, Pooling and Fully Connected Layers! Medium. https://medium.com/@prajeeshprathap/the-secret-to-understanding-cnns-convolution-feature-maps-pooling-and-fully-connected-layers-97055431a847

Rodríguez, E. (2023, 7 octubre). La Importancia de la Función de Pérdida en Machine Learning. Canal Innova.

https://canalinnova.com/la-importancia-de-la-funcion-de-perdida-en-machine-learning/

Verdeguer, J. (2020). Redes neuronales para la clasificación y segmentación de imágenes médicas [Tesis de maestría, Universitat Politècnica de València]. Recuperado 16 de mayo de 2024, de https://m.riunet.upv.es/bitstream/handle/10251/158739/Verdeguer%20-%20Redes%20neuronales%20para%20la%20clasificación%20y%20segmentación%20de%20imágenes%20médicas.pdf?sequence=1&isAllowed=y#:~:text=Las%20redes%20densas%20son%20las,neurona%20de%20la%20capa%20anterior.

Venturi, Luca (2020). Hands-On Vision and Behavior for Self-Driving Cars. Packt Publishing.