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 Fernández

Avance 2
Ingeniería de Características
(FE - *Feature Engineering)*

“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: 12 de mayo de 2024

Nuestros datos son un conjunto de imágenes de espectrogramas de ruido las cuales pertenecen a una de las dos clases: aceptables (1 - ok) o rechazadas (0 - nok). Como se mencionó en los avances anteriores, este conjunto de datos no contiene valores faltantes o atípicos, ni se cuenta con alguna otra variable. En este caso, para poder lograr un entrenamiento exitoso de nuestro modelo se requieren los siguientes procedimientos.

- Recorte de imágenes y transformación a escala de grises
- Renombramiento de imágenes
- Normalización del tamaño de las imágenes en píxeles
- Data augmentation

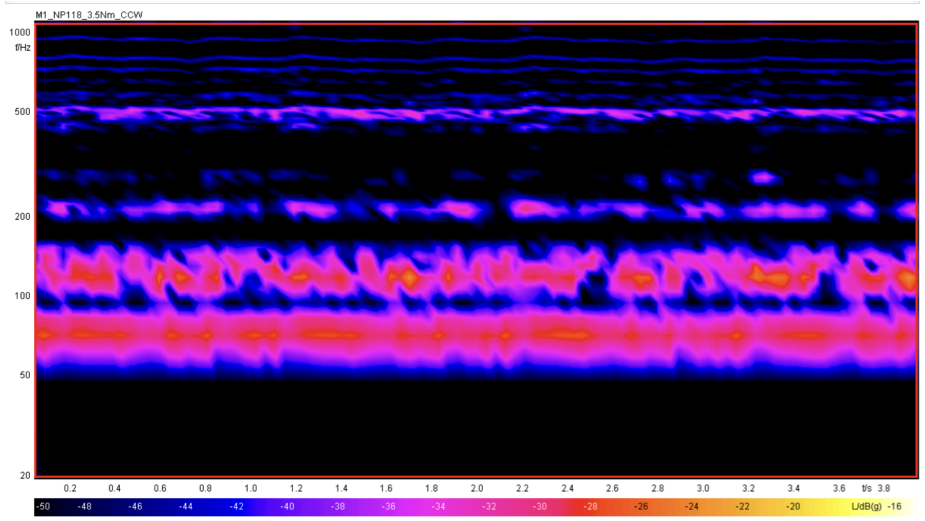
Construcción

Recorte de imágenes y transformación a escala de grises

Para está actividad se utilizó la librería de PIL de Python para poder recortar nuestra ROI (Region of interest), ya que las imágenes que nos fueron brindadas contienen ejes que muestran los valores de las frecuencias en Y y la intensidad del color en X. Dado que todas las gráficas tienen la misma escala, se optó por recortarlas. Cabe mencionar que para entrenar y utilizar modelos más adelante, esa "información"extra en las imágenes no nos es útil.

Figura 1.

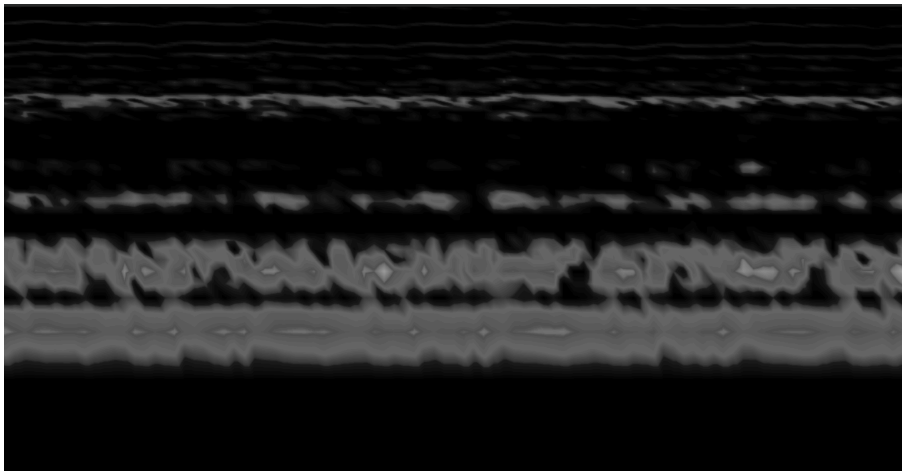
Imagen Original



Nota. Ejemplo de imagen original donde se muestran las intensidades del sonido y sus frecuencias. el cuadro rojo muestra la zona de imágen que se preservó, la ROI a utilizar.

Posteriormente al recorte de la imagen se convirtió a escala de grises para poder trabajar solamente con un canal en nuestros modelos. Teniendo nuestra imagen final:

Figura 2.
Imagen final



Nota. Aquí se muestra un ejemplo de cómo quedan las imágenes finales con su recorte del marco y la escala de grises aplicada.

A continuación se muestra el código de las funciones que se utilizaron:

```python
import os
from PIL import Image

def convert_to_grayscale(input_directory, output_directory):
    """
    Convert all images in the specified input directory to grayscale and save them in the output directory.

    Parameters:
        input_directory (str): Path to the directory containing the input image files.
        output_directory (str): Path to the directory where grayscale images will be saved.
    """
    # Create the output directory if it doesn't exist
    if not os.path.exists(output_directory):
        os.makedirs(output_directory)

    # List all files in the input directory
    files = os.listdir(input_directory)

    # Filter out only image files
    image_files = [file for file in files if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp'))]

    # Convert each image file to grayscale
    for image_file in image_files:
        input_image_path = os.path.join(input_directory, image_file)
        output_image_path = os.path.join(output_directory, image_file)
        convert_image_to_grayscale(input_image_path, output_image_path)
```

```python
def convert_image_to_grayscale(input_image_path, output_image_path):
    """
    Convert an image to grayscale.

    Parameters:
        input_image_path (str): Path to the input image file.
        output_image_path (str): Path to save the grayscale image.
    """
    # Open the input image
    img = Image.open(input_image_path)

    # Convert the image to grayscale
    grayscale_img = img.convert("L")

    # Save the grayscale image
    grayscale_img.save(output_image_path)
```

Normalización

Renombramiento y normalización de imágenes

Con el fin de tener un mejor orden en los datos se optó por renombrarlos cada imagen como OK y NOK, para que después de manera automática con Python, se lea el nombre de cada imagen para determinar la categoría de cada archivo y crear un DataFrame de pandas con esta información.

```
[4]:
for i, image in enumerate(images):
  original_path = os.path.join(folder_input_nok, image) #ruta completa de ca

  new_label = f'nok.({i+1})' + os.path.splitext(image)[1]

  new_path = os.path.join(folder_output_nok, new_label)

  os.rename(original_path,new_path)

  print(f'Renombrado {image} a {new_label} y movido a NOK_etiquetadas_500')

drive.flush_and_unmount()
print('Etiquetado completado')
```

```
filenames = os.listdir("Drives")
categories = []
for filename in filenames:
    category = filename.split('.')
    if category == 'ok':
        categories.append(1)
    else:
        categories.append(0)

df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})
df.head()
```

Out[4]:

	filename	category
0	.ipynb_checkpoints	0
1	nok.(1).png	0
2	nok.(10).png	0
3	nok.(100).png	0
4	nok.(101).png	0

Posteriormente a esto se normalizaron los tamaños de cada imagen a 224 x 224 píxeles para posteriormente realizar data augmentation.

```
from keras.models import Sequential
from keras import layers
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense, Activation,GlobalMaxPooling2D
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.applications import VGG16
from keras.models import Model

image_size = 224
input_shape = (image_size, image_size, 3)

epochs = 5
batch_size = 16

pre_trained_model = VGG16(input_shape=input_shape, include_top=False, weights="imagenet")
```

Data augmentation

Para generar más imágenes a nuestro set de datos, se utilizó la función de 'Image data generator' de la biblioteca de Keras, Estas configuraciones permiten aumentar artificialmente el tamaño del conjunto de datos de entrenamiento mediante la aplicación de transformaciones aleatorias a las imágenes de entrada. El aumento de datos ayuda al modelo a generalizar mejor y a mejorar su capacidad para reconocer patrones en nuevas imágenes.

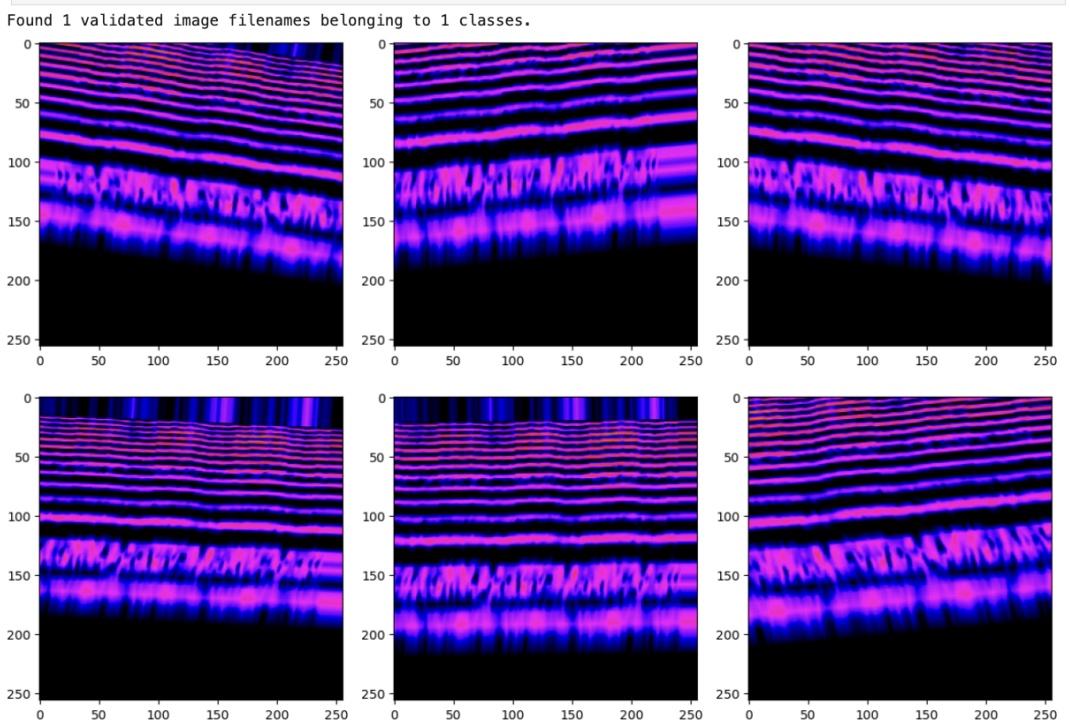
```
train_datagen = ImageDataGenerator(
    rotation_range=15,
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    width_shift_range=0.1,
    height_shift_range=0.1
)

train_generator = train_datagen.flow_from_dataframe(
    train_df,
    "Drives/",
    x_col='filename',
    y_col='category',
    class_mode='binary',
    target_size=(image_size, image_size),
    batch_size=batch_size
)
```

Función de para generación de datos de imágenes

Figura 3.

Imágenes obtenidas por aumentación de datos.



Nota. Estas nuevas imágenes generadas pertenecen a una sola imagen de espectrograma del conjunto de datos original.

Conclusiones

El preprocesamiento de imágenes y la normalización, junto con la aumentación de datos es fundamental en el entrenamiento de modelos de aprendizaje profundo en visión por computadora. El recorte enfoca la atención del modelo y estandariza el tamaño de las imágenes. La conversión a escala de grises simplifica el procesamiento y reduce la dimensionalidad. La normalización mejora la convergencia y la robustez al contraste. La aumentación de datos aumenta la cantidad y variabilidad de los datos de entrenamiento, mejorando la generalización y la resistencia a pequeñas variaciones. En conjunto, estas técnicas permiten desarrollar modelos más precisos y robustos en una variedad de aplicaciones de visión por computadora y de clasificación de las imágenes por redes neuronales convolucionales. Para los siguientes avances del proyecto se pretende crear conjuntos de datos diferentes mezclando estas técnicas de preprocesamiento y normalización de las imágenes con el objetivo de probar diferentes conjuntos de datos con diferentes modelos de clasificación para así compararlos con métricas de desempeño y seleccionar el modelo que arroje los mejores resultados.

Repositorio de GitHub

Link del repositorio en Github: https://github.com/emm-gl/project_mna

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.

Francois Chollet. (2017). Deep Learning with Python. Manning.